VisualDSP++® 4.0
Getting Started Guide

Revision 1.0, January 2005

Part Number
82-000420-01

Analog Devices, Inc.
One Technology Way
Norwood, Mass. 02062-9106

ANALOG DEVICES

Copyright Information

© 2005 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, the CROSSCORE logo, VisualDSP++, Blackfin, SHARC, TigerSHARC, and EZ-KIT Lite are registered trademarks of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

CONTENTS

PREFACE

Purpose of This Manual vii
Intended Audience vii
Manual Contents viii
What's New in This Manual viii
Technical or Customer Support ix
Supported Processors x
Product Information xi
MyAnalog.com xi
Processor Product Information xi
Related Documents xii
Online Technical Documentation xiii
Accessing Documentation From VisualDSP++ xiv
Accessing Documentation From Windows xiv
Accessing Documentation From the Web xv
Printed Manuals xv
VisualAudio or VisualDSP++ Documentation Set xv
Hardware Tools Manuals xv
Processor Manuals xv

Data Sheets xvi
Contacting DSP Publications xvi
Notation Conventions xvi

FEATURES AND TOOLS

VisualDSP++ Features 1-1
New Features in Release 4.0 1-5
Code Development Tools 1-7

BASIC TUTORIAL

Overview 2-1
Exercise One: Building and Running a C Program 2-3
Step 1: Start VisualDSP++ and Open a Project 2-4
Step 2: Build the dotprodc Project 2-7
Step 3: Run the Program 2-10
Step 4: Run dotprodc 2-15
Exercise Two: Modifying a C Program to Call an Assembly Routine 2-16
Step 1: Create a New Project 2-16
Step 2: Add Source Files to dot_product_asm 2-22
Step 3: Create a Linker Description File 2-23
Step 4: Modify the Project Source Files 2-28
Step 5: Use the Expert Linker to Modify dot_prod_asm.ldf 2-32
Step 6: Rebuild and Run dot_product_asm 2-35

Exercise Three: Plotting Data 2-37
Step 1: Load the FIR Program 2-37
Step 2: Open a Plot Window 2-39
Step 3: Run the FIR Program and View the Data 2-42
Exercise Four: Linear Profiling 2-51
Step 1: Load the FIR Program 2-51
Step 2: Open the Profiling Window 2-52
Step 3: Collect and Examine the Linear Profile Data 2-54

ADVANCED TUTORIAL

Overview 3-1
Exercise One: Using Profile-Guided Optimization 3-2
Step 1: Load the Project 3-4
Step 2: Configure a Data Set 3-5
Step 3: Attach an Input Stream 3-10
Step 4: Configure Additional Data Sets 3-15
Step 5: Create PGO Files and Optimize the Program 3-17
Step 6: Compare Execution Times 3-18
Exercise Two: Using Background Telemetry Channel 3-22
Adding BTC to Your DSP Application 3-22
Running the BTC Assembly Demo 3-24
Step 1: Load the BTC_AsmDemo Project 3-25
Step 2: Examine the BTC Commands 3-26
Step 3: Set Up the BTC Memory Window and View Data 3-29

Running the BTC FFT Demo 3-37
Step 1: Build the FFT Demo 3-38
Step 2: Plot BTC Data 3-39
Step 3: Record and Analyze BTC Data 3-44

INDEX

PREFACE

Thank you for purchasing VisualDSP++™, the development software for Analog Devices processors.

Purpose of This Manual

The *VisualDSP++ 4.0 Getting Started Guide* provides basic and advanced tutorials that highlight many VisualDSP++ features. By completing the step-by-step procedures, you will become familiar with the VisualDSP++ environment and learn how to use these features in your own digital signal processing (DSP) development projects.

Intended Audience

This manual is intended for DSP programmers who are familiar with Analog Devices processors. The manual assumes that the audience has a working knowledge of Analog Devices processor architecture and instruction set.

DSP programmers who are unfamiliar with Analog Devices processors should refer to their processor's Hardware Reference and Instruction Set Reference, which describe the processor architecture and instruction set. Note that the *ADSP-BF533 Blackfin Processor Hardware Reference* includes information about the ADSP-BF531 and ADSP-BF532 processors.

Manual Contents

This manual consists of:

- Chapter 1, “Features and Tools”

  Provides an overview of VisualDSP++ features and code development tools

- Chapter 2, “Basic Tutorial”

  Provides step-by-step instructions for creating sessions, and for building and debugging projects by using examples of C/C++ and assembly sources

  The tutorial is organized to follow the steps that you take in developing a typical programming project. Before you begin actual programming, you should be familiar with the architecture of your particular processor and the other software development tools.

- Chapter 3, “Advanced Tutorial”

  Provides step-by-step instructions for using profile-guided optimization (PGO) and background telemetry channel (BTC)

What’s New in This Manual

This manual contains updated example screens and procedures for the 4.0 Integrated Development and Debugging Environment (IDDE).

Technical or Customer Support

You can reach Analog Devices, Inc. Customer Support in the following ways:

- Visit the Embedded Processing and DSP products Web site at
  `http://www.analog.com/processors/technicalSupport`
- E-mail tools questions to
  `dsptools.support@analog.com`
- E-mail processor questions to
  `dsp.support@analog.com`
- Phone questions to **1-800-ANALOGD**
- Contact your Analog Devices, Inc. local sales office or authorized distributor
- Send questions by mail to:

```
Analog Devices, Inc.
One Technology Way
P.O. Box 9106
Norwood, MA 02062-9106
USA
```

Supported Processors

The name "*Blackfin*®" refers to a family of Analog Devices 16-bit, embedded processors. VisualDSP++ currently supports the following Blackfin processors.

ADSP-BF531	ADSP-BF532 (formerly ADSP-21532)
ADSP-BF533	ADSP-BF535 (formerly ADSP-21535)
ADSP-BF534	ADSP-BF536
ADSP-BF537	ADSP-BF538
ADSP-BF539	ADSP-BF566
ADSP-BF561	AD6532

The names "*SHARC*®" and "*TigerSHARC*®" refer to the family of Analog Devices 32-bit, digital signal processors. VisualDSP++ currently supports the following SHARC processors.

ADSP-21020	ADSP-21261
ADSP-21060	ADSP-21262
ADSP-21061	ADSP-21266
ADSP-21062	ADSP-21267
ADSP-21065L	ADSP-21363
ADSP-21160	ADSP-21364
ADSP-21161	ADSP-21365

VisualDSP++ currently supports the following TigerSHARC processors.

ADSP-TS101	ADSP-TS202
ADSP-TS201	ADSP-TS203

Product Information

You can obtain product information from the Analog Devices website, from the product CD-ROM, or from the printed publications (manuals).

Analog Devices is online at www.analog.com. Our website provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

MyAnalog.com

MyAnalog.com is a free feature of the Analog Devices website that allows customization of a webpage to display only the latest information on products you are interested in. You can also choose to receive weekly email notification containing updates to the webpages that meet your interests. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Registration:

Visit www.myanalog.com to sign up. Click **Register** to use MyAnalog.com. Registration takes about five minutes and serves as means for you to select the information you want to receive.

If you are already a registered user, just log on. Your user name is your email address.

Processor Product Information

For information on embedded processors and DSPs, visit our Web site at www.analog.com/processors, which provides access to technical publications, data sheets, application notes, product overviews, and product announcements.

You may also obtain additional information about Analog Devices and its products in any of the following ways.

- E-mail questions or requests for information to
  `embedded.support@analog.com`
- Fax questions or requests for information to
  1-781-461-3010 (North America)
  089/76 903-557 (Europe)
- Access the FTP Web site at
  `ftp ftp.analog.com` or `ftp 137.71.23.21`
  `ftp://ftp.analog.com`

Related Documents

For information on product related development software, see these publications:

- *VisualDSP++ 4.0 User's Guide*
- *VisualDSP++ 4.0 Assembler and Preprocessor Manual*
- *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for SHARC Processors*
- *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for TigerSHARC Processors*
- *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for Blackfin Processors*
- *VisualDSP++ 4.0 Linker and Utilities Manual*
- *VisualDSP++ 4.0 Loader Manual*
- *VisualDSP++ 4.0 Product Release Bulletin*

- *VisualDSP++ 4.0 Kernel (VDK) User's Guide*
- *VisualDSP++ 4.0 Quick Installation Reference Card*

For hardware information, refer to your processors's hardware reference, programming reference, or data sheet. All documentation is available online. Most documentation is available in printed form.

Visit the Technical Library Web site to access all processor and tools manuals and data sheets:

```
http://www.analog.com/processors/resources/technicalLibrary
```

Online Technical Documentation

Online documentation includes the VisualDSP++ Help system, software tools manuals, hardware tools manuals, processor manuals, Dinkum Abridged C++ library, and Flexible License Manager (FlexLM) network license manager software documentation. You can easily search across the entire VisualDSP++ documentation set for any topic of interest using the Search function of VisualDSP++ Help system. For easy printing, supplementary `.PDF` files of most manuals are also provided.

Each documentation file type is described as follows.

File	Description
`.CHM`	Help system files and manuals in Help format
`.HTM` or `.HTML`	Dinkum Abridged C++ library and FlexLM network license manager software documentation. Viewing and printing the `.HTML` files requires a browser, such as Internet Explorer 4.0 (or higher).
`.PDF`	VisualDSP++ and processor manuals in Portable Documentation Format (PDF). Viewing and printing the `.PDF` files requires a PDF reader, such as Adobe Acrobat Reader (4.0 or higher).

Access the online documentation from the VisualDSP++ environment, Windows® Explorer, or the Analog Devices Web site.

If documentation is not installed on your system as part of the software installation, you can add it from the VisualDSP++ CD-ROM at any time by running the VisualDSP++ installation. Access the online documentation from the VisualDSP++ environment, Windows® Explorer, or the Analog Devices Web site.

Accessing Documentation From VisualDSP++

From the VisualDSP++ environment:

- Access VisualDSP++ online Help from the Help menu's **Contents**, **Search**, and **Index** commands.
- Open online Help from context-sensitive user interface items (toolbar buttons, menu commands, and windows).

Accessing Documentation From Windows

In addition to any shortcuts you may have constructed, there are other ways to open VisualDSP++ online Help or the supplementary documentation from Windows.

Help system files (`.CHM`) are located in the `Help` folder, and `.PDF` files are located in the `Docs` folder of your VisualDSP++ installation CD-ROM.

Using Windows Explorer

- Double-click the `vdsp-help.chm` file, which is the master Help system, to access all the other `.CHM` files.
- Double-click any file that is part of the VisualDSP++ documentation set.

Accessing Documentation From the Web

Download manuals at the following Web site:
`http://www.analog.com/processors/resources/technicalLibrary/manuals`

Select a processor family and book title. Download archive (`.ZIP`) files, one for each manual. Use any archive management software, such as WinZip, to decompress downloaded files.

Printed Manuals

For general questions regarding literature ordering, call the Literature Center at **1-800-ANALOGD** (**1-800-262-5643**) and follow the prompts.

VisualAudio or VisualDSP++ Documentation Set

To purchase VisualDSP++ manuals, call **1-603-883-2430**. The manuals may be purchased only as a kit.

If you do not have an account with Analog Devices, you are referred to Analog Devices distributors. For information on our distributors, log onto `http://www.analog.com/salesdir/continent.asp`.

Hardware Tools Manuals

To purchase EZ-KIT Lite® and In-Circuit Emulator (ICE) manuals, call **1-603-883-2430**. The manuals may be ordered by title or by product number located on the back cover of each manual.

Processor Manuals

Hardware reference and instruction set reference manuals may be ordered through the Literature Center at **1-800-ANALOGD** (**1-800-262-5643**), or downloaded from the Analog Devices Web site. Manuals may be ordered by title or by product number located on the back cover of each manual.

Data Sheets

All data sheets (preliminary and production) may be downloaded from the Analog Devices Web site. Only production (final) data sheets (Rev. 0, A, B, C, and so on) can be obtained from the Literature Center at **1-800-ANALOGD** (**1-800-262-5643**); they also can be downloaded from the Web site.

To have a data sheet faxed to you, call the Analog Devices Faxback System at **1-800-446-6212**. Follow the prompts and a list of data sheet code numbers will be faxed to you. If the data sheet you want is not listed, check for it on the Web site.

Contacting DSP Publications

Please send your comments and recommendations for improving our manuals and online Help. You can contact us by sending an email to `dsp.techpubs@analog.com`:

Notation Conventions

Text conventions used in this manual are identified and described as follows.

Example	Description
Close command (**File** menu)	Titles in reference sections indicate the location of an item within the VisualAudio environment's menu system (for example, the **Close** command appears on the **File** menu).
`{this | that}`	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. One or the other is required.
`[this | that]`	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional `this` or `that`.

Example	Description
`[this,…]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `letter gothic` font.
`filename`	Non-keyword placeholders appear in text with italic style format.
	Note: For correct operation, ... A Note provides supplementary information on a related topic. In the online version of this book, the word **Note** appears instead of this symbol.
	Caution: Incorrect device operation may result if ... **Caution:** Device damage may result if ... A Caution identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word **Caution** appears instead of this symbol.
	Warning: Injury to device users may result if ... A Warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word **Warning** appears instead of this symbol.

Additional conventions, which apply only to specific chapters, may appear throughout this document.

1 FEATURES AND TOOLS

This chapter contains the following topics.

- "VisualDSP++ Features" on page 1-1
- "New Features in Release 4.0" on page 1-5
- "Code Development Tools" on page 1-6

VisualDSP++ Features

VisualDSP++ provides these features:

- **Extensive editing capabilities.** Create and modify source files by using multiple language syntax highlighting, drag-and-drop, bookmarks, and other standard editing operations. View files generated by the code development tools.
- **Flexible project management.** Specify a project definition that identifies the files, dependencies, and tools that you use to build projects. Create this project definition once or modify it to meet changing development needs.

- **Easy access to code development tools.** Analog Devices provides these code development tools: C/C++ compiler, assembler, linker, splitter, and loader. Specify options for these tools by using dialog boxes instead of complicated command-line scripts. Options that control how the tools process inputs and generate outputs have a one-to-one correspondence to command-line switches. Define options for a single file or for an entire project. Define these options once or modify them as necessary.
- **Flexible project build options.** Control builds at the file or project level. VisualDSP++ enables you to build files or projects selectively, update project dependencies, or incrementally build only the files that have changed since the previous build. View the status of your project build in progress. If the build reports an error, double-click on the file name in the error message to open that source file. Then correct the error, rebuild the file or project, and start a debug session.
- **VisualDSP++ Kernel (VDK) support**. Add VDK support to a project to structure and scale application development. The **Kernel** page of the **Project** window enables you to manipulate events, event bits, priorities, semaphores, and thread types.
- **Flexible workspace management.** Create up to ten workspaces and quickly switch between them. Assigning a different project to each workspace enables you to build and debug multiple projects in a single session.
- **Easy movement between debug and build activities.** Start the debug session and move freely between editing, build, and debug activities.

Figure 1-1 shows the Integrated Development and Debugging Environment (IDDE).

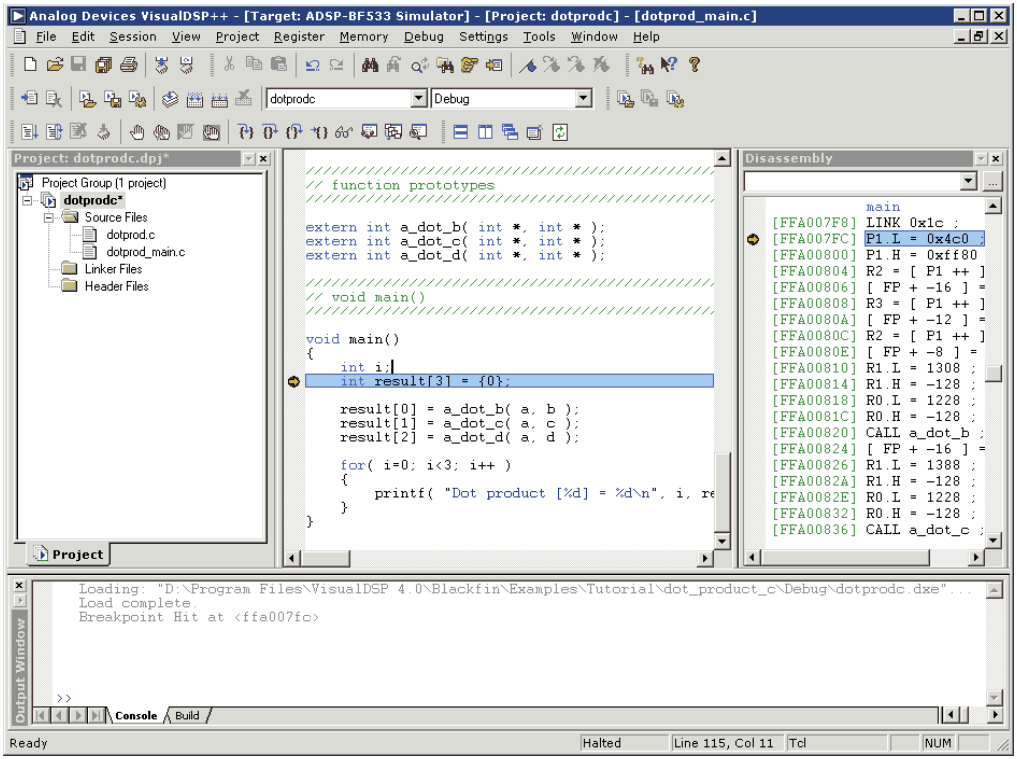

Figure 1-1. The VisualDSP++ IDDE

VisualDSP++ reduces debugging time by providing these key features:

- **Easy-to-use debugging activities.** Debug with one common, easy-to-use interface for all processor simulators and emulators, or hardware evaluation and development boards. Switch easily between these targets.
- **Multiple language support.** Debug programs written in C, C++, or assembly, and view your program in machine code. For programs written in C/C++, you can view the source in C/C++ or mixed C/C++ and assembly, and display the values of local variables or evaluate expressions (global and local) based on the current context.
- **Effective debug control.** Set breakpoints on symbols and addresses and then step through the program's execution to find problems in coding logic. Set watchpoints (conditional breakpoints) on registers, stacks, and memory locations to identify when they are accessed.
- **Tools for improving performance.** Use the trace, profile, and linear and statistical profiles to identify bottlenecks in your DSP application and to identify program optimization needs. Use plotting to view data arrays graphically. Generate interrupts, outputs, and inputs to simulate real-world application conditions.

New Features in Release 4.0

Release 4.0 includes these new features and enhancements:

- **Support for new processors**. VisualDSP++ supports these new Blackfin processors: ADSP-BF534, ADSP-BF536, ADSP-BF537, ADSP-BF538 ADSP-BF539, ADSP-BF566
- **New Project Wizard**. This feature lets you create and configure a DSP project with code that sets up peripherals like cache, DMA ports, I/O ports, and so on. An LDF can be added to the project to link in only necessary libraries, and plug-ins can be added to analyze system performance and determine optimal parameters for the peripherals. Selecting **New** and then **Project** from the **File** menu now launches a **New Project Wizard** where you specify project options such as **Name**, **Directory**, **Project type** (standard application or multithreaded application using VDK), **Processor**, **Silicon revision**, **Output type** (executable file or Loader file).
- **Build engine enhancements**. You can select multiple files in the **Project** window and issue a **Build Files** command to build multiple files simultaneously. You can use a **Batch Build** command to open a dialog box that lets you manage batch builds for multiple build targets. In the **Output** window, you now have the option of displaying only file names instead of command lines while building a project. A new Project Group file lets you store project group information in a file to allow the quick restoring and convenient sharing of information. The **Project** menu includes new project group options (**Open Project Group**, **Save Project Group**, **Save Project Group As**, and **Close Project Group**).

- **Changes to Image Viewer**. In addition to Grayscale 8 and RGB24, these pixel formats are now supported: Grayscale 12, Grayscale 16, RGB555, RGB565, and RGB32. These YUV pixel formats are also supported: YUV2 (4:2:2), YVYU (4:2:2), UYVY (4:2:2), and IYU2 (4:4:4). Display and processing speed has been improved through the use of the graphics device interface (GDI+). Image Viewer can now convert color formats from YUV to RGB and visa-versa, and transfer JPEG binary data to and from DSP memory. New functions have been added for advanced image processing (rotating and flipping images, filtering, auto-refreshing). New controls have been added to the **Image Configuration** dialog box, a toolbar has been added to the **Image Viewer** window.
- **New Project Options dialog box**. A tree navigation control on the left side of the dialog box has replaced tabs for selecting the page view (**General**, **Compiler**, **Assemble**, **Link**, **Load**, **Pre-build**, or **Post-build**). Other changes include new fields and pages for configuring project options.
- **Changes to licensing management**. Three types of licenses are available: TST (evaluation), KIT (evaluation), and ADI (permanent).

  A TST (test drive) license provides unlimited (unrestricted) access to VisualDSP++ for emulation and simulation. You must register the software to receive a TST serial number, which expires 90 days after installation. After the 90 days, the software is inoperable.

  A KIT license provides 10 days to register and validate the installation. Once validated, the KIT licence extends to 90 (ninety) days of a full evaluation in total. At the end of the evaluation period, unless you have added a permanent license, simulator and emulator connections become prohibited and the size of the user program is limited.

An ADI license provides 30 (thirty) days to register and validate the installation for a permanent use. If you fail to register and validate your installation during the 30-day evaluation period, your copy of VisualDSP++ 4.0 becomes inoperable.

Code Development Tools

Code development tools include:

- C/C++ compiler
- Run-time library with over 100 math, DSP, and C run-time library routines
- Assembler
- Linker
- Splitter
- Loader
- Simulator
- Emulator (must be purchased separately from VisualDSP++)

These tools enable you to develop applications that take full advantage of your processor's architecture.

The VisualDSP++ linker supports multiprocessing, shared memory, and memory overlays.

The code development tools provide these key features:

- **Easy-to-program C, C++, and assembly languages.** Program in C/C++, assembly, or a mix of C/C++ and assembly in one source. The assembly language is based on an algebraic syntax that is easy to learn, program, and debug.
- **Flexible system definition.** Define multiple types of executables for a single type of processor in one Linker Description File (`.LDF`). Specify input files, including objects, libraries, shared memory files, overlay files, and executables.
- **Support for overlays, multiprocessors, and shared memory executables.** The linker places code and resolves symbols in multi-processor memory space for use by multiprocessor systems. The loader enables you to configure multiple processors with less code and faster boot time. Create host, link port, and PROM boot images.

Software and hardware tool kits include context-sensitive Help and manuals in PDF format.

For details about assembly syntax, refer to the *VisualDSP++ 4.0 Assembler and Preprocessor Manual* for your target processor.

2 BASIC TUTORIAL

This chapter contains the following topics.

- "Overview" on page 2-1
- "Exercise One: Building and Running a C Program" on page 2-3
- "Exercise Two: Modifying a C Program to Call an Assembly Routine" on page 2-16
- "Exercise Three: Plotting Data" on page 2-37
- "Exercise Four: Linear Profiling" on page 2-51

Overview

The Basic Tutorial demonstrates key features and capabilities of the VisualDSP++ Integrated Development and Debugging Environment (IDDE). The exercises use sample programs written in C and assembly for Blackfin processors.

You can use different Blackfin processors with only minor changes to the Linker Description Files (`.LDFs`) included with each project. VisualDSP++ includes basic Linker Description Files for each processor type in the `ldf` folder. For Blackfin processors, the folder's default installation path is:

```
Program Files\Analog Devices\VisualDSP 4.0\Blackfin\ldf
```

The source files for these exercises are installed during the VisualDSP++ software installation.

The tutorial contains four exercises:

- In **Exercise One**, you start up VisualDSP++, build a project containing C source code, and profile the performance of a C function.
- In **Exercise Two**, you create a new project, create a Linker Description File to link with the assembly routine, rebuild the project, and profile the performance of the assembly language routine.
- In **Exercise Three**, you plot the various waveforms produced by a Finite Impulse Response (FIR) algorithm.
- In **Exercise Four**, you use linear profiling to examine the efficiency of the FIR algorithm used in Exercise Three. Using the collected linear profile data, you pinpoint the most time-consuming areas of the algorithm, which are likely to require hand tuning in the assembly language.

The ADSP-BF53x Family Simulator and ADSP-BF533 processor are used for all exercises.

Tip: Become familiar with the VisualDSP++ toolbar buttons, shown in Figure 2-1. They are shortcuts for menu commands such as **Open** a file and **Run** a program. Toolbar buttons and menu commands that are not available for tasks are disabled and displayed in gray.

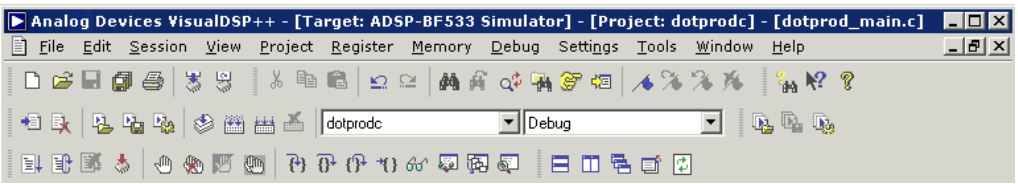
Analog Devices VisualDSP++ - [Target: ADSP-BF533 Simulator] - [Project: dotprodc] - [dotprod_main.c]
File Edit Session View Project Register Memory Debug Settings Tools Window Help
dotprodc
Debug

Figure 2-1. VisualDSP++ Toolbar Buttons

VisualDSP++ is a licensed software product. To run the software, you must have a valid license installed on your system. If you try to run VisualDSP++ and a license is not installed, a message window opens to let you add a license. For details about license management, see the *VisualDSP++ 4.0 User's Guide* or VisualDSP++ online Help.

Exercise One: Building and Running a C Program

In this exercise, you:

- Start up the VisualDSP++ environment
- Open and build an existing project
- Examine windows and dialog boxes
- Run the program

The sources for this exercise are in the `dot_product_c` folder. The default installation path is:

```
Program Files\Analog Devices\VisualDSP 4.0\Blackfin\Examples\
Tutorial\dot_product_c
```

Step 1: Start VisualDSP++ and Open a Project

To start VisualDSP++ and open a project:

1. Click the Windows **Start** button and select **Programs**, **Analog Devices**, **VisualDSP++ 4.0**, and **VisualDSP++ Environment**.

   If you are running VisualDSP++ for the first time, the **New Session** dialog box (Figure 2-6 on page 2-11) opens to enable you to set up a session.

   a. Select the values shown in Table 2-1.

Table 2-1. Session Specification

Box	Value
Debug Target	ADSP-BF5xx Blackfin Family Simulator
Platform	ADSP-BF5xx Single Processor Simulator
Session Name	ADSP-BF533 Simulator
Processor	ADSP-BF533

   b. Click **OK**. The VisualDSP++ main window appears.

   If you have already run VisualDSP++ and the **Reload last project at startup** option is selected on the **Project** page under **Settings** and **Preferences**, VisualDSP++ opens the last project that you worked on. To close this project, choose **Close** from the **Project** menu, and then click **No** when prompted to save the project. Since you have made no changes to the project, you do not have to save it.

2. From the **File** menu, choose **Open** and then **Project**.

   VisualDSP++ displays the **Open Project** dialog box.

3. In the **Look in** box, open the `Program Files\Analog Devices` folder and double-click the following subfolders in succession.

   `VisualDSP 4.0\Blackfin\Examples\Tutorial\dot_product_c`

This path is based on the default installation.

4. Double-click the `dotprodc` project (`.dpj`) file.

   VisualDSP++ loads the project in the **Project** window, as shown in Figure 2-2. The environment displays messages in the **Output** window as it processes the project settings and file dependencies.

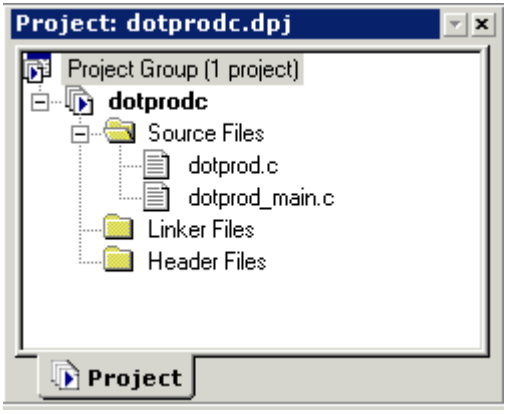

Figure 2-2. Project Loaded in the Project Window

The `dotprodc` project comprises two C language source files, `dotprod.c` and `dotprod_main.c`, which define the arrays and calculate their dot products.

5. From the **Settings** menu, choose **Preferences** to open the **Preferences** dialog box, shown in Figure 2-3.

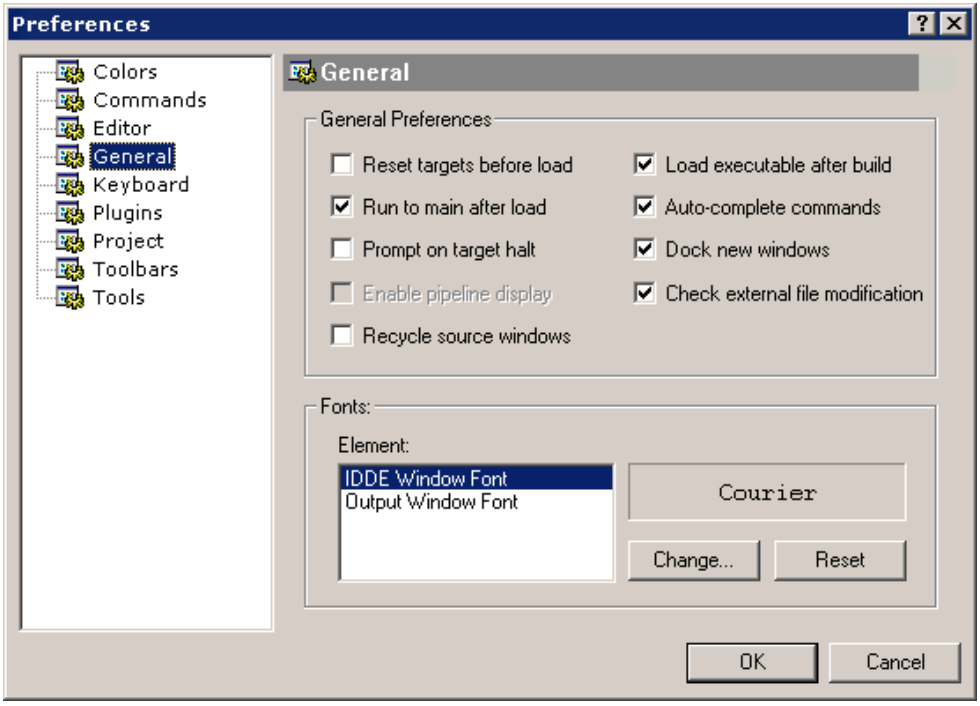

Figure 2-3. Preferences Dialog Box

6. On the **General** page, under **General Preferences**, ensure that the following options are selected.
   - **Run to main after load**
   - **Load executable after build**
7. Click **OK** to close the **Preferences** dialog box.

   The VisualDSP++ main window appears. You are now ready to build the project.

Step 2: Build the dotprodc Project

To build the `dotprodc` project:

1. From the **Project** menu, choose **Build Project**.

   VisualDSP++ first checks and updates the project dependencies and then builds the project by using the project source files.

   As the build progresses, the **Output** window displays status messages (error and informational) from the tools. For example, when a tool detects invalid syntax or a missing reference, the tool reports the error in the **Output** window.

   If you double-click the file name in the error message, VisualDSP++ opens the source file in an editor window. You can then edit the source to correct the error, rebuild, and launch the debug session. If the project build is up-to-date (the files, dependencies, and options have not changed since the last project build), no build is performed unless you run the **Rebuild All** command. Instead, you see the message "`Project is up to date`." If the build has no errors, a message reports "`Build completed successfully`."

In this example (Figure 2-4) notice that the compiler detects an undefined identifier and issues the following error in the **Build** view of the **Output** window.

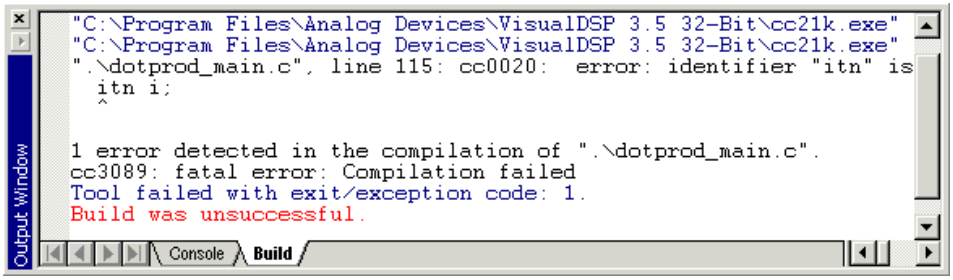

Figure 2-4. Example of Error Message

2. Double-click the error message text in the **Output** window.

   VisualDSP++ opens the C source file `dotprod_main.c` in an editor window and places the cursor on the line that contains the error (see Figure 2-5).

   The editor window in Figure 2-5 shows that the integer variable declaration `int` has been misspelled as `itn`.

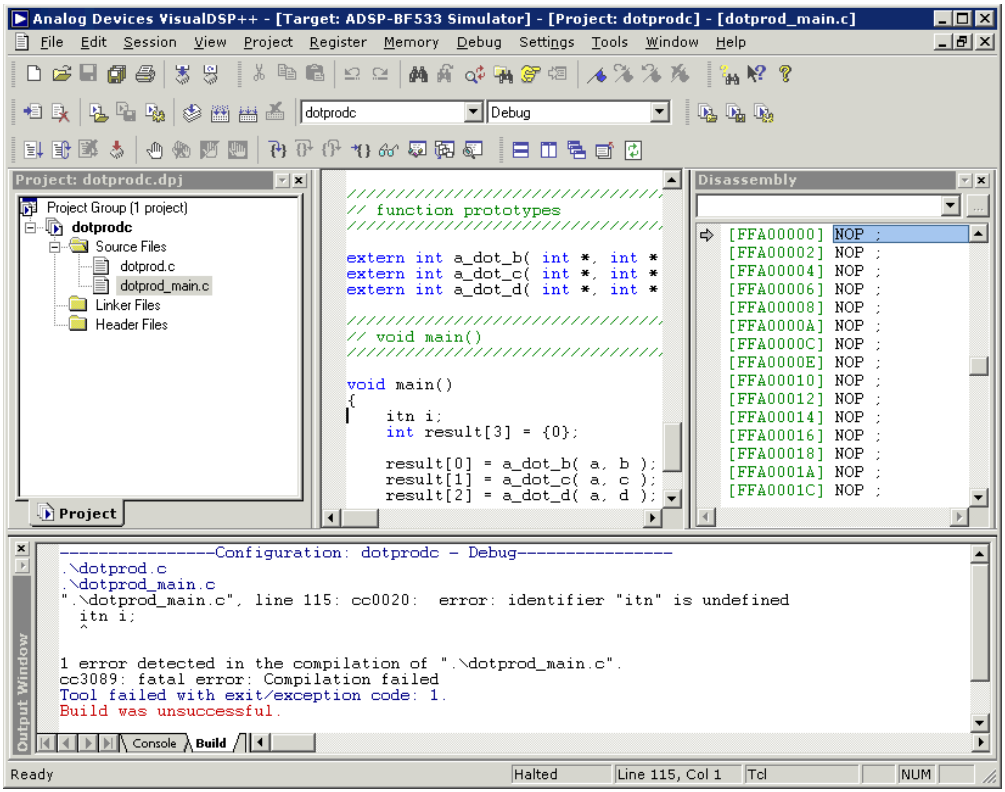

Figure 2-5. Output Window and Editor Window

3. In the editor window, click on `itn` and change it to `int`. Notice that `int` is now color coded to signify that it is a valid C keyword.

4. Save the source file by choosing **File dotprod_main.c** from the **File→Save** menu.

5. Build the project again by choosing **Build Project** from the **Project** menu. The project is now built without any errors, as reported in the **Build** view of the **Output** window.

Now that you have built your project successfully, you can run the example program.

Step 3: Run the Program

In this procedure, you:

- Set up the debug session before running the program
- View debugger windows and dialog boxes

Since you enabled **Load executable after build** on the **General** page in the **Preferences** dialog box, the executable file `dotprodc.dxe` is automatically downloaded to the target. If the debug session's processor does not match the project's build target, VisualDSP++ reports this discrepancy and asks if you want to select another session before downloading the executable to the target. If VisualDSP++ does not open the **Session List** dialog box, skip steps 1–4.

To set up the debug session:

1. In the **Session List** dialog box, click **New Session** to open the **New Session** dialog box, shown in Figure 2-6.

   For subsequent debugging sessions, use the **New Session** command on the **Sessions** menu to open the **New Session** dialog box.

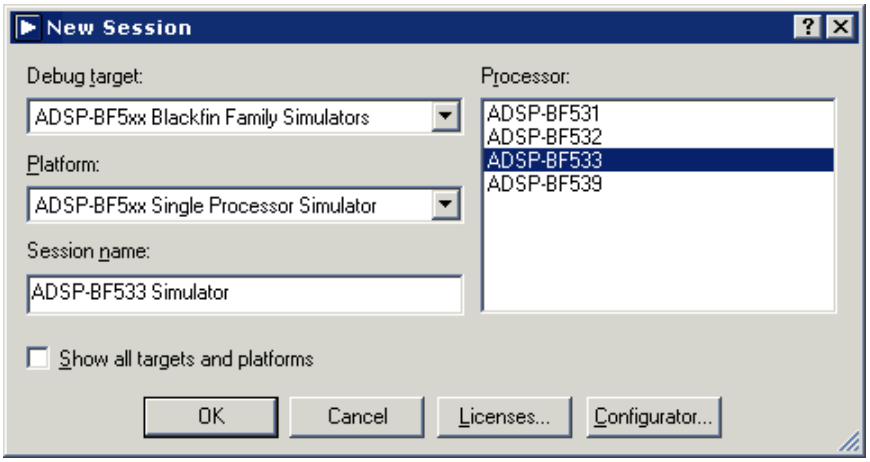

Figure 2-6. New Session Dialog Box

2. Specify the target and processor information listed in Table 2-2.

Table 2-2. Session Specification

Box	Value
Debug Target	ADSP-BF5xx Blackfin Family Simulator
Platform	ADSP-BF5xx Single Processor Simulator
Session Name	ADSP-BF533 Simulator
Processor	ADSP-BF533

3. Click **OK** to close the **New Session** dialog box and return to the **Session List** dialog box.
4. With the new session name highlighted, click **Activate**.

If you do not click **Activate**, the session mismatch message appears again.

VisualDSP++ closes the **Session List** dialog box, automatically loads your project's executable file (`dotprodc.dxe`), and advances to the main function of your code (see Figure 2-7).

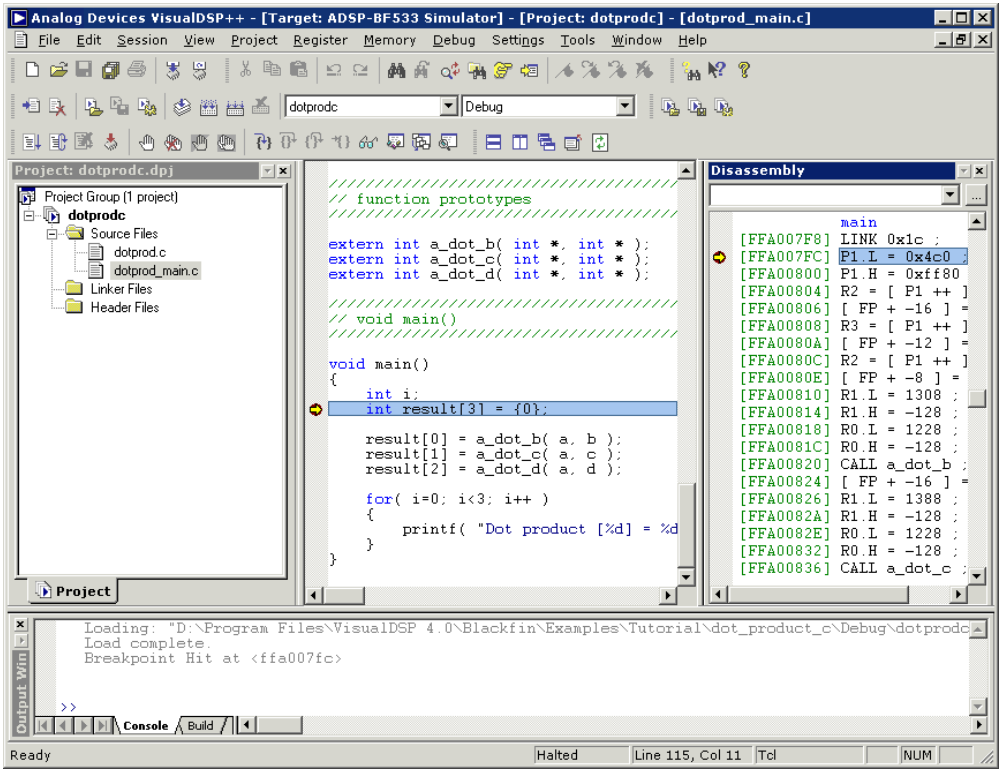

Figure 2-7. Loading dotprodc.dxe

5. Look at the information in the open windows.

   The **Output** window's **Console** page contains messages about the status of the debug session. In this case, VisualDSP++ reports that the `dotprodc.dxe` load is complete.

   The **Disassembly** window displays the assembly code for the executable. Use the scroll bars to move around the **Disassembly** window.

   Note that a solid red circle and a yellow arrow appear at the start of the program labeled "`main`". The solid red circle (●) indicates that a breakpoint is set on that instruction, and the yellow arrow (➪) indicates that the processor is currently halted at that instruction. When VisualDSP++ loads your C program, it automatically sets two breakpoints, one at the beginning and one at the end of code execution. Your breakpoint locations may differ slightly from those shown in the examples in this book.

6. From the **Settings** menu, choose **Breakpoints** to view the breakpoints set in your program. VisualDSP++ displays the **Breakpoints** dialog box, shown in Figure 2-8.

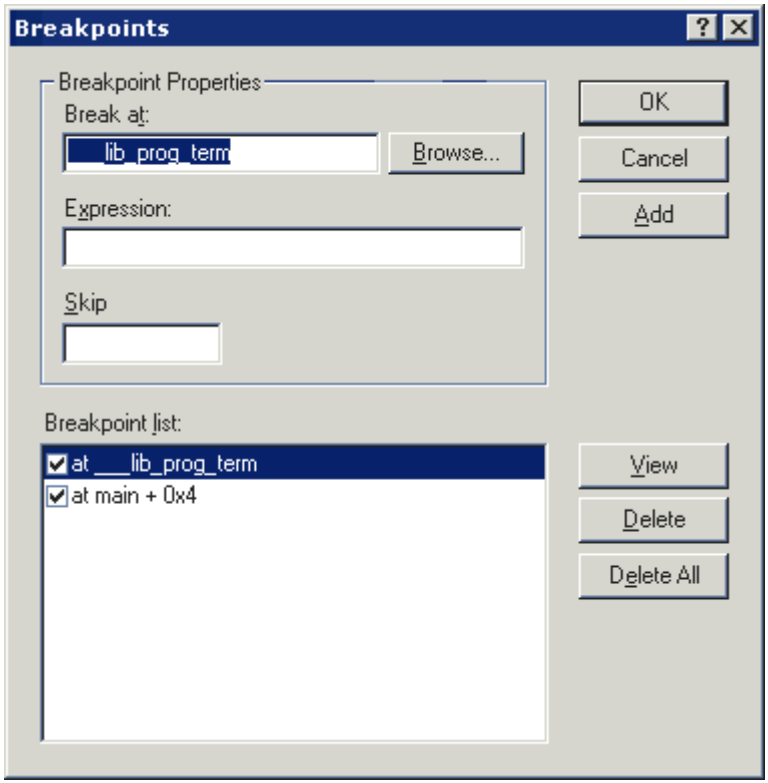

Figure 2-8. Breakpoints Dialog Box

The breakpoints are set at these C program locations:

- `at main + 0x04`
- `at __lib_prog_term`

The **Breakpoints** dialog box enables you to view, add, and delete breakpoints and to browse for symbols. In the **Disassembly** and editor windows, double-clicking on a line of code toggles (adds or deletes) breakpoints. In the editor window, however, you must place the mouse pointer in the gutter before double-clicking.

These tool buttons set or clear breakpoints:

Toggles a breakpoint for the current line

Clears all breakpoints

7. Click **OK** or **Cancel** to exit the **Breakpoints** dialog box.

Step 4: Run dotprodc

To run `dotprodc`, click the **Run** button or choose **Run** from the **Debug** menu.

VisualDSP++ computes the dot products and displays the following results on the **Console** view (Figure 2-9) of the **Output** window.

```
Dot product [0] = 13273595
Dot product [1] = -49956078
Dot product [2] = 35872518
```

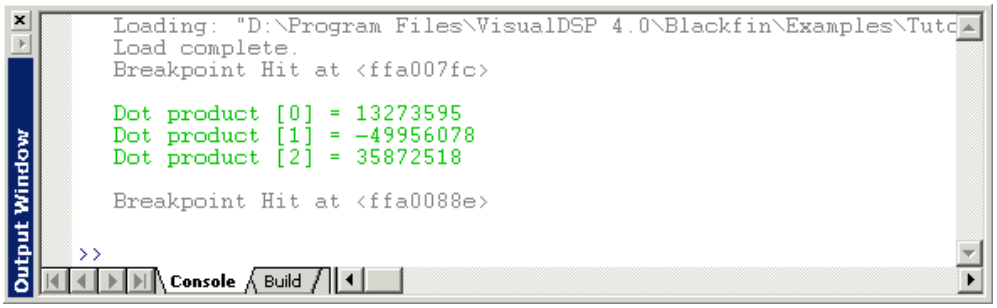

Figure 2-9. Results of the dotprodc Program

You are now ready to begin Exercise Two.

Exercise Two: Modifying a C Program to Call an Assembly Routine

In Exercise One, you built and ran a C program. In this exercise, you:

- Modify the C program to call an assembly language routine
- Create a Linker Description File to link with the assembly routine
- Rebuild the project

The project files are largely identical to those of Exercise One. Minor modifications illustrate the changes needed to call an assembly language routine from C source code.

Step 1: Create a New Project

To create a new project:

1. From the **File** menu, choose **Close** and then **Project dotprodc** to close the `dotprodc` project.

   Click **Yes** when prompted to close all open source windows.

   If you have modified your project during this session, you are prompted to save the project. Click **No**.

2. From the **File** menu, choose **New** and then **Project** to open the **Project Wizard**, shown in Figure 2-10.

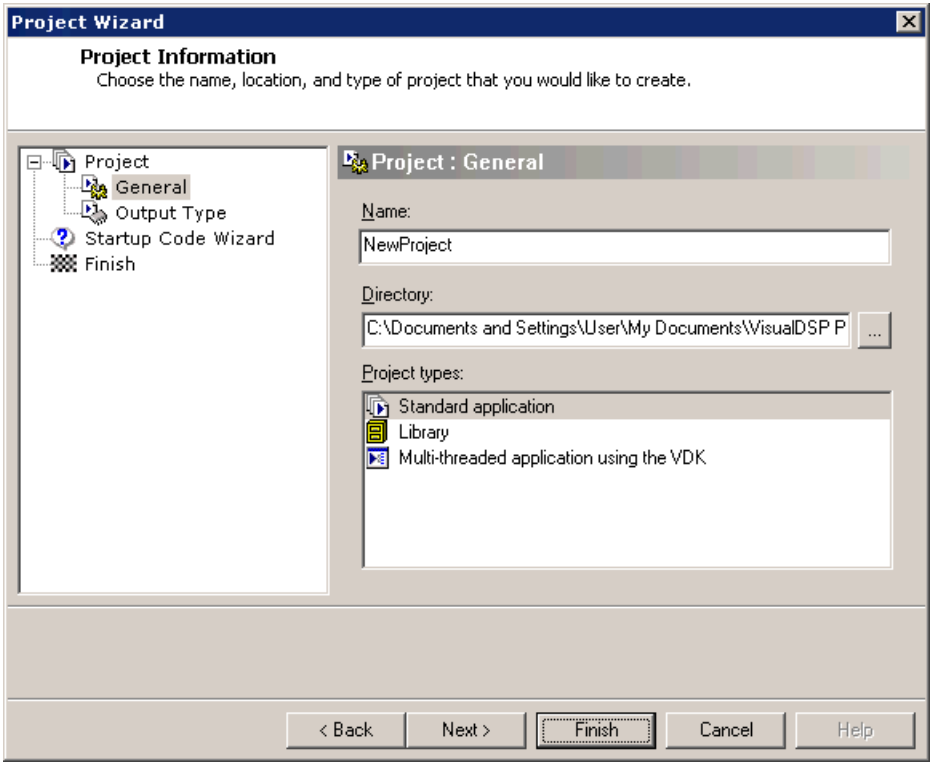

Figure 2-10. Project Wizard – General Page

3. Click the browse button ... to the right of the **Directory** field to open the **Browse For Folder** dialog box. Locate the `dot_product_asm` tutorial folder and click **OK**. By default this directory is in the following location.

```
Program Files\Analog Devices\VisualDSP 4.0\Blackfin\
Examples\Tutorial\dot_product_asm
```

4. In the **Name** field, type `dot_product_asm` and click **Finish**. The new project is created and is shown in the **Project** window of the IDDE.
5. From the **Project** menu click the **Project Options** command to display the **Project Options** dialog box (Figure 2-11).

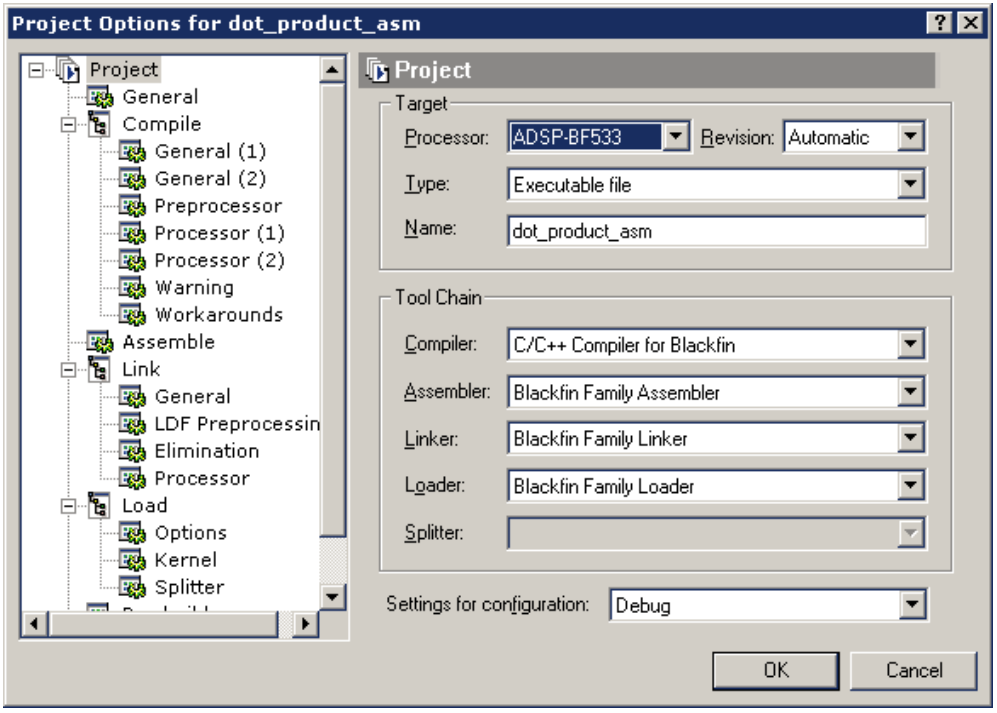

Figure 2-11. Project Options Dialog Box – Project Page

This dialog box enables you to specify project build information.

6. Take a moment to view the various pages in the **Project Options** dialog box by selecting them from the tree on the left: **Project**, **General**, **Compile**, **Assemble**, **Link**, **Load**, **Pre-Build**, and **Post-Build**. On each page, you specify the tool options used to build the project.

7. On the **Project** page (Figure 2-11), verify that the values shown in Table 2-3 are entered here.

Table 2-3. Completing the Project Page

Field	Value
Processor	ADSP-BF533
Revision	Automatic
Type	Executable file
Name	dot_product_asm
Settings for configuration	Debug

These settings specify information for building an executable file for the ADSP-BF533 processor. The executable contains debug information, so you can examine program execution.

8. Click the **Compile** tab to display the **Compile** page, shown in Figure 2-12.

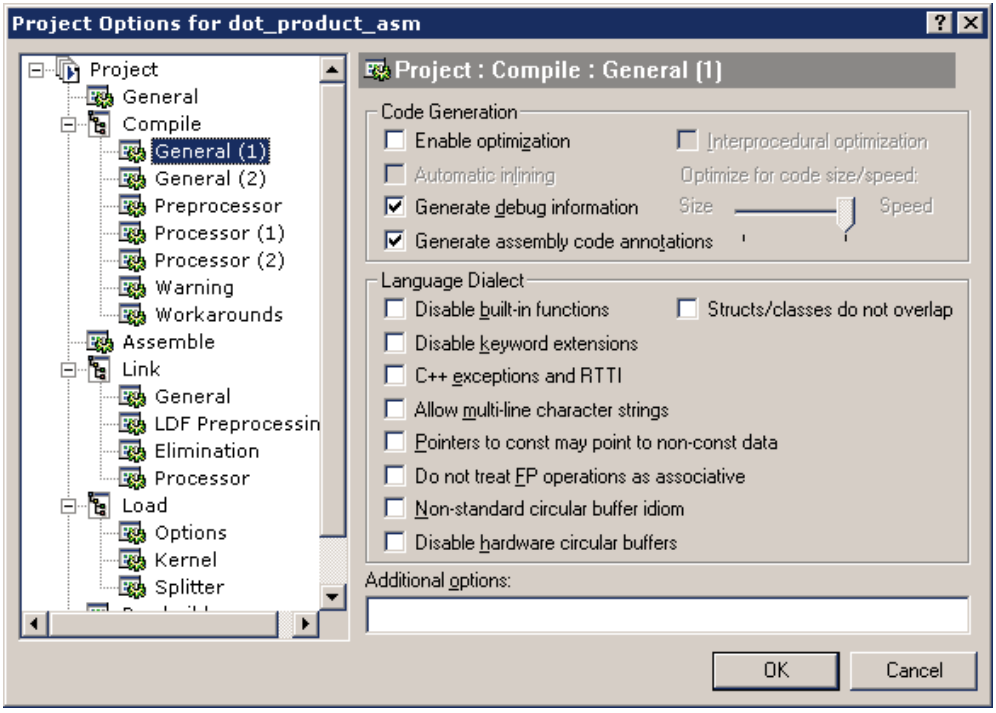

Figure 2-12. Project Options Dialog Box – Compile Page

9. Specify these settings in the **Code Generation** group box:

    a. Select the **Enable optimization** check box to enable optimization.

    b. Select the **Generate debug information** check box, if it is not already selected, to enable debug information for the C source.

These settings direct the C compiler to optimize code for the ADSP-BF533 processor. Because the optimization takes advantage of DSP architecture and assembly language features, some of the C debug information is not saved. Therefore, debugging is performed through debug information at the assembly language level.

10. Click **OK** to apply changes to the project options and to close the **Project Options** dialog box. When prompted to add support for the VisualDSP++ kernel, click **No**.

You are now ready to add the source files to the project.

Step 2: Add Source Files to dot_product_asm

To add the source files to the new project:

1. Click the **Add File** button , or from the **Project** menu, choose **Add to Project**, and then choose **File(s)**.

   The **Add Files** dialog box (Figure 2-13) appears.

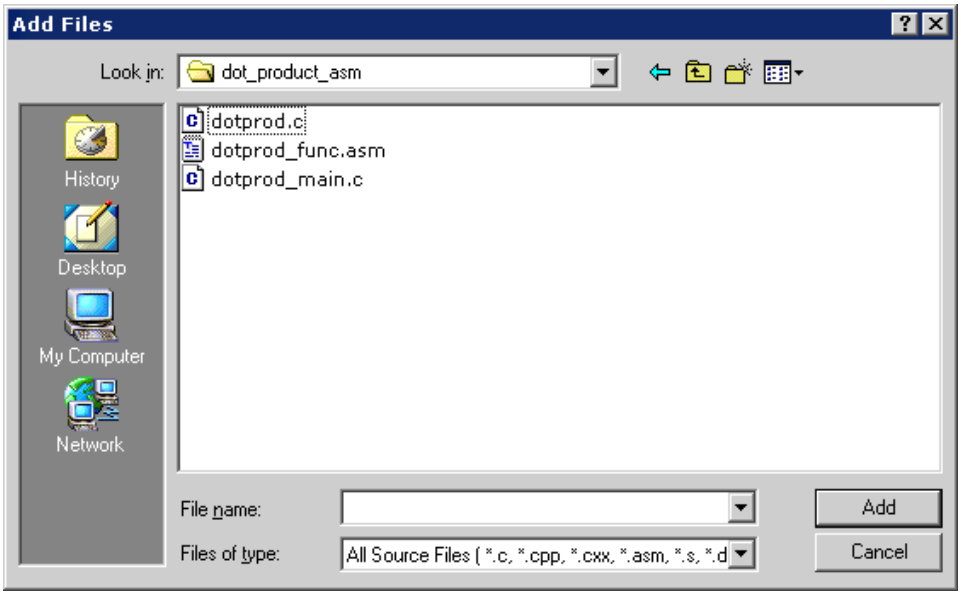

Figure 2-13. Add Files Dialog Box – Adding Source Files to the Project

2. In the **Look in** box, locate the project folder, `dot_product_asm`.
3. In the **Files of type** box, select **All Source Files** from the drop-down list.
4. Hold down the **Ctrl** key and click `dotprod.c` and `dotprod_main.c`. Then click **Add**.

To display the files that you added in step 4, open the `Source Files` folder in the **Project** window.

You are now ready to create a Linker Description File for the project.

Step 3: Create a Linker Description File

In this procedure, you use the Expert Linker to create a Linker Description File for the project.

To create a Linker Description File:

1. From the **Tools** menu, choose **Expert Linker** and then choose **Create LDF** to open the **Create LDF Wizard**, shown in Figure 2-14.

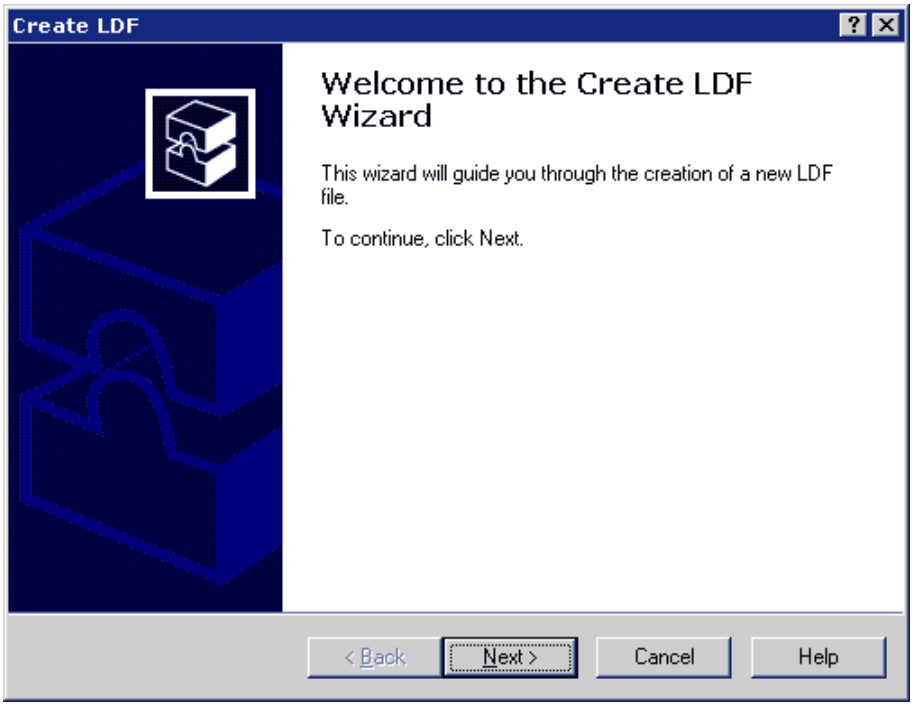
Create LDF
Welcome to the Create LDF Wizard
This wizard will guide you through the creation of a new LDF file.
To continue, click Next.
< Back
Next >
Cancel
Help

Figure 2-14. Create LDF Wizard

2. Click **Next** to display the **Create LDF – Step 1 of 3** page, shown in Figure 2-15.

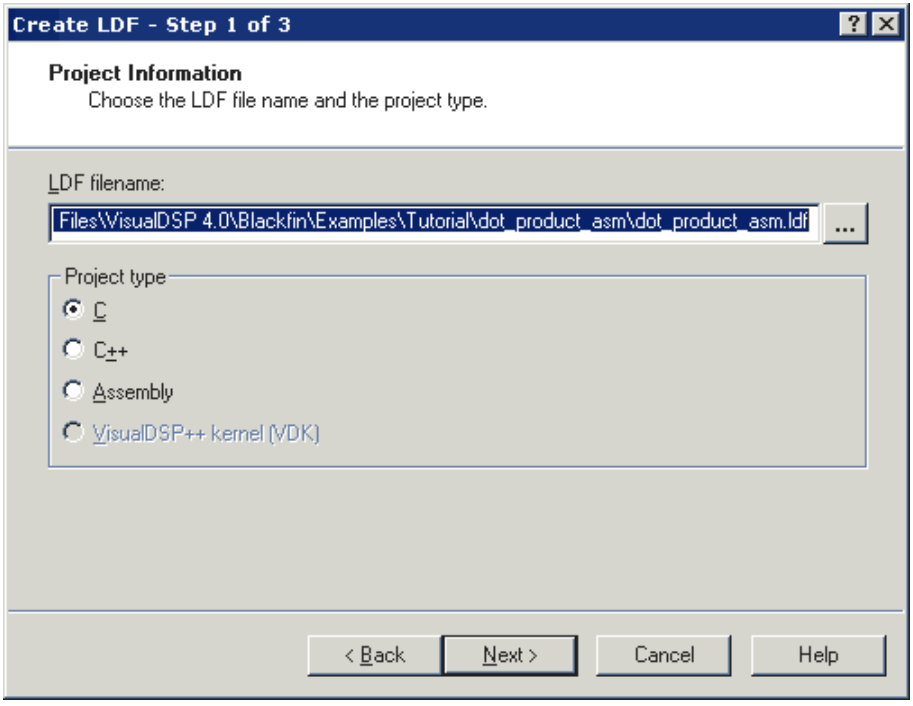

Figure 2-15. Create LDF – Step 1 of 3 Page

This page enables you to assign the **LDF file name** (based on the project name) and to select the **Project type**.

3. Accept the values selected for your project and click **Next** to display the **Create LDF – Step 2 of 3** page, shown in Figure 2-16.

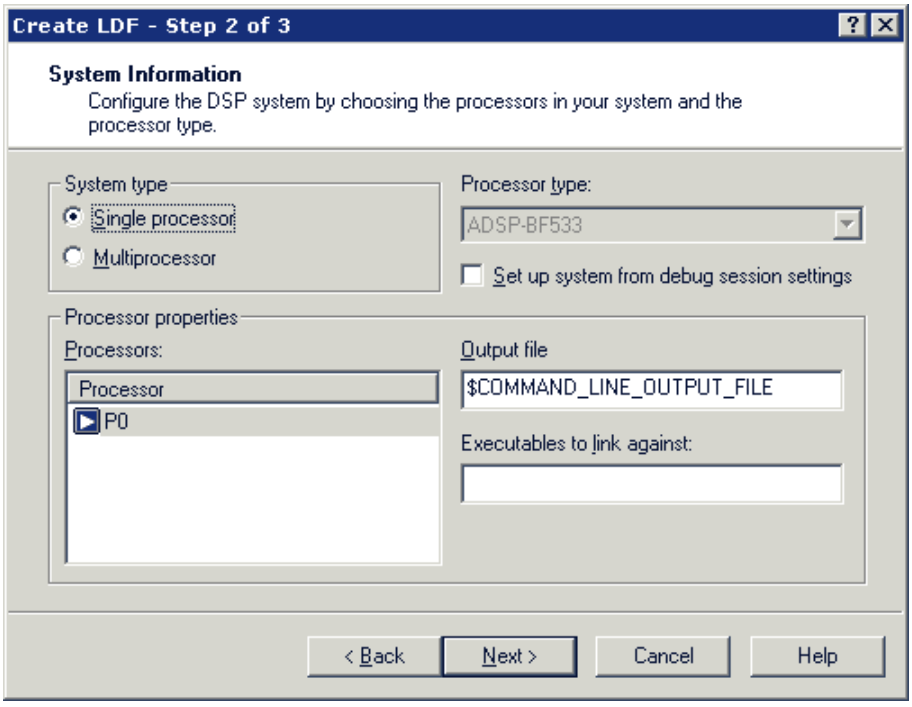

Figure 2-16. Create LDF – Step 2 of 3 Page

This page enables you to set the **System type** (defaulted to **Single processor**), the **Processor type** (defaulted to **ADSP-BF533** to match the project), and the name of the linker **Output file** (defaulted to the name selected by the project).

4. Accept the default values and click **Next** to display the **Create LDF – Step 3 of 3** page, shown in Figure 2-17.

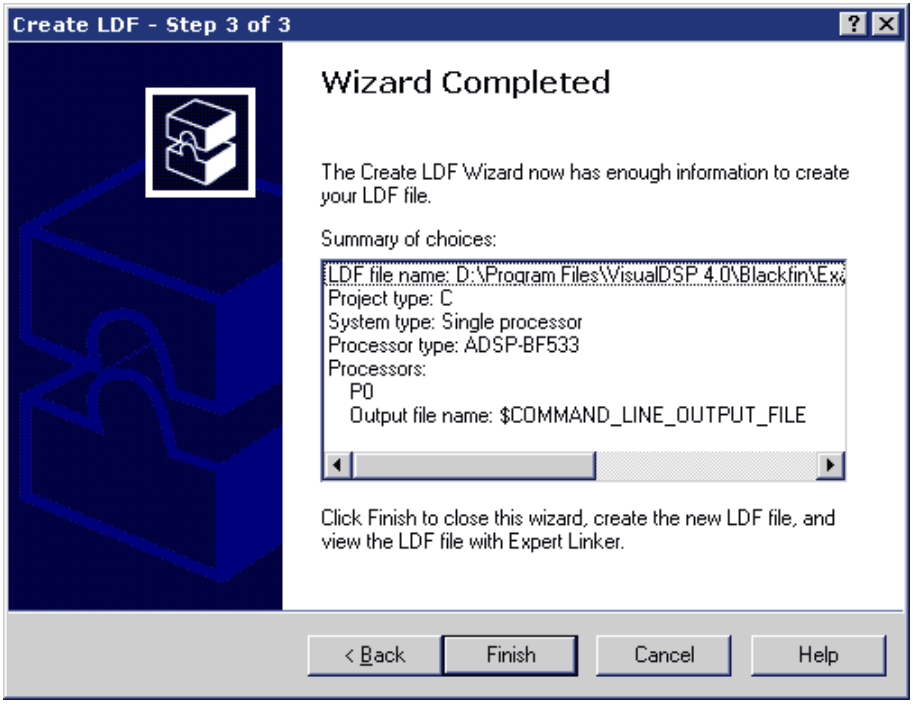
Create LDF - Step 3 of 3
Wizard Completed
The Create LDF Wizard now has enough information to create your LDF file.
Summary of choices:
LDF file name: D:\Program Files\VisualDSP 4.0\Blackfin\Ex
Project type: C
System type: Single processor
Processor type: ADSP-BF533
Processors:
P0
Output file name: $COMMAND_LINE_OUTPUT_FILE
Click Finish to close this wizard, create the new LDF file, and view the LDF file with Expert Linker.
< Back
Finish
Cancel
Help

Figure 2-17. Create LDF – Step 3 of 3 Page

5. Review the **Summary of choices** and click **Finish** to create the `.LDF` file.

   You now have a new `.LDF` file in your project. The new file is in the **Linker Files** folder in the **Project** window.

   The **Expert Linker** window opens with a representation of the new `.LDF` file. This file is complete for this project. Close the **Expert Linker** window.

6. Click the **Rebuild All** button to build the project. The C source file opens in an editor window, and execution halts.

The C version of the project is now complete. You are now ready to modify the sources to call the assembly function.

Step 4: Modify the Project Source Files

In this procedure, you:

- Modify `dotprod_main.c` to call `a_dot_c_asm` instead of `a_dot_c`
- Save the modified file

To modify `dotprod_main.c` to call the assembly function:

1. Resize or maximize the editor window for better viewing.
2. From the **Edit** menu, choose **Find** to open the **Find** dialog box, shown in Figure 2-18.

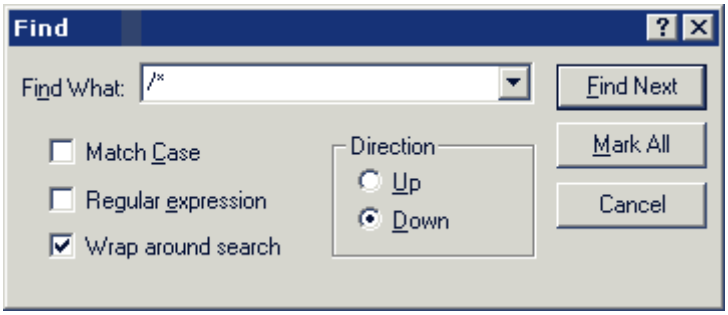

Figure 2-18. Find Dialog Box: Locating Occurrences of /*

3. In the **Find What** box, type `/*`, and then click **Mark All**.

   The editor bookmarks all lines containing `/*` and positions the cursor at the first instance of `/*` in the `extern int a_dot_c_asm` declaration.

4. Select the comment characters `/*` and use the **Ctrl+X** key combination to cut the comment characters from the beginning of the `a_dot_c_asm` declaration. Then move the cursor up one line and use the **Ctrl+V** key combination to paste the comment characters at the beginning of the `a_dot_c` declaration. Because syntax coloring is turned on, the code changes color as you cut and paste the comment characters.

   Repeat this step for the end-of-comment characters `*/` at the end of the `a_dot_c_asm` declaration. The `a_dot_c` declaration is now fully commented out, and the `a_dot_c_asm` declaration is no longer commented.

5. Press **F2** to move to the next bookmark.

   The editor positions the cursor on the `/*` in the function call to `a_dot_c_asm`, which is currently commented out. Note that the previous line is the function call to the `a_dot_c` routine.

6. Press **Ctrl+X** to cut the comment characters from the beginning of the function call to `a_dot_c_asm`. Then move the cursor up one line and press **Ctrl+V** to paste the comment characters at the beginning of the call to `a_dot_c`.

   Repeat this step for the end-of-comment characters `*/`. The `main()` function is now calling the `a_dot_c_asm` routine instead of the `a_dot_c` function (previously called in Exercise One).

   Figure 2-19 shows the changes made in step 6.

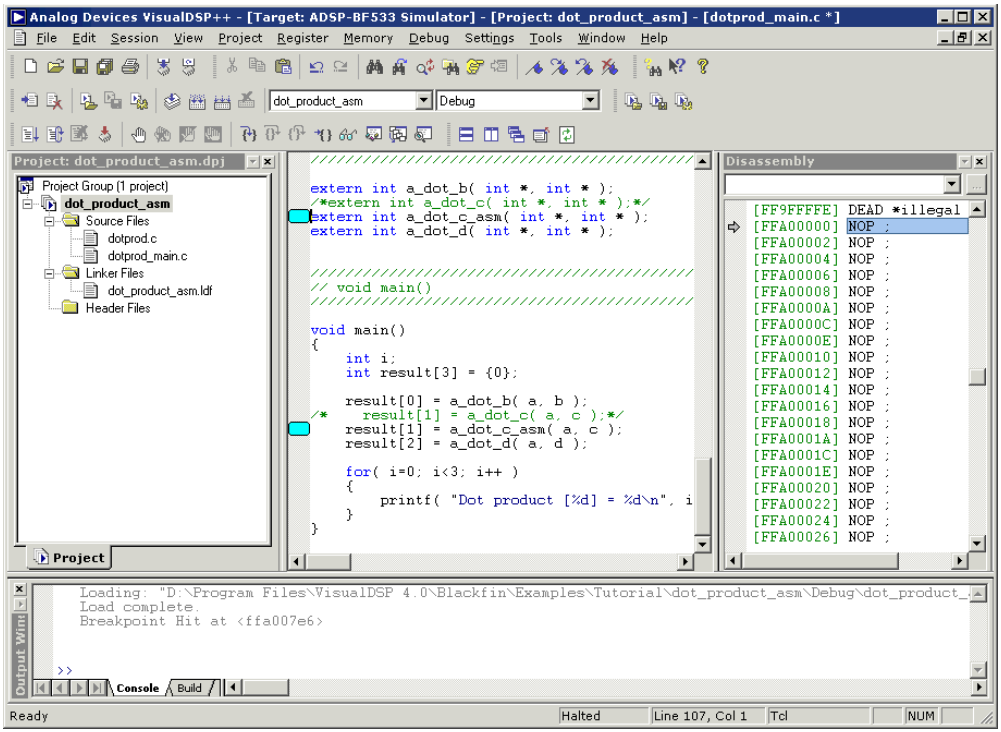

Figure 2-19. Modifying dotprod_main.c to Call a_dot_c_asm

7. From the **File** menu, choose **Save** and then **File dotprod_main.c** to save the changes to the file.

8. Place the cursor in the editor window. Then, from the **File** menu, choose **Close** and then **File dotprod_main.c** to close the `dotprod_main.c` file.

You are now ready to modify `dotprodasm.ldf`.

Step 5: Use the Expert Linker to Modify dot_prod_asm.ldf

In this procedure you:

- View the Expert Linker representation of the `.LDF` file that you created
- Modify the `.LDF` file to map in the section for the `a_dot_c_asm` assembly routine

To examine and then modify `dot_prod_asm.ldf` to link with the assembly function:

1. Click the **Add File** button .
2. Select `dotprod_func.asm` and click **Add**.
3. Build the project by performing one of these actions:
   - Click the **Build Project** button .
   - From the **Project** menu, choose **Build Project**.
4. Notice the linker error in the **Output** window (Figure 2-20).

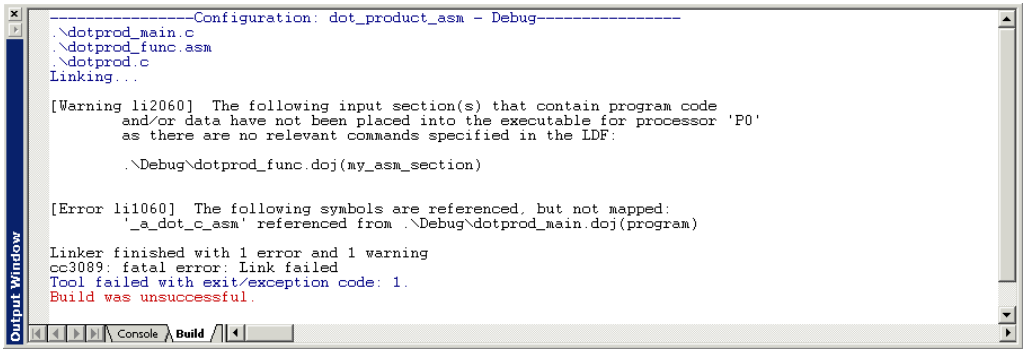
----------Configuration: dot_product_asm - Debug----------
.\dotprod_main.c
.\dotprod_func.asm
.\dotprod.c
Linking...
[Warning li2060] The following input section(s) that contain program code
and/or data have not been placed into the executable for processor 'P0'
as there are no relevant commands specified in the LDF:
.\Debug\dotprod_func.doj(my_asm_section)
[Error li1060] The following symbols are referenced, but not mapped:
'_a_dot_c_asm' referenced from .\Debug\dotprod_main.doj(program)
Linker finished with 1 error and 1 warning
cc3089: fatal error: Link failed
Tool failed with exit/exception code: 1.
Build was unsuccessful.
Output Window
Console
Build

Figure 2-20. Output Window: Linker Error

5. In the **Project** window, double-click in the `dot_prod_asm.ldf` file. The **Expert Linker** window (Figure 2-21) opens with a graphical representation of your file.

   Resize the window to expand the view and change the view mode. To display the tree view shown in Figure 2-21, right-click in the right pane, choose **View Mode**, and then choose **Memory Map Tree**.

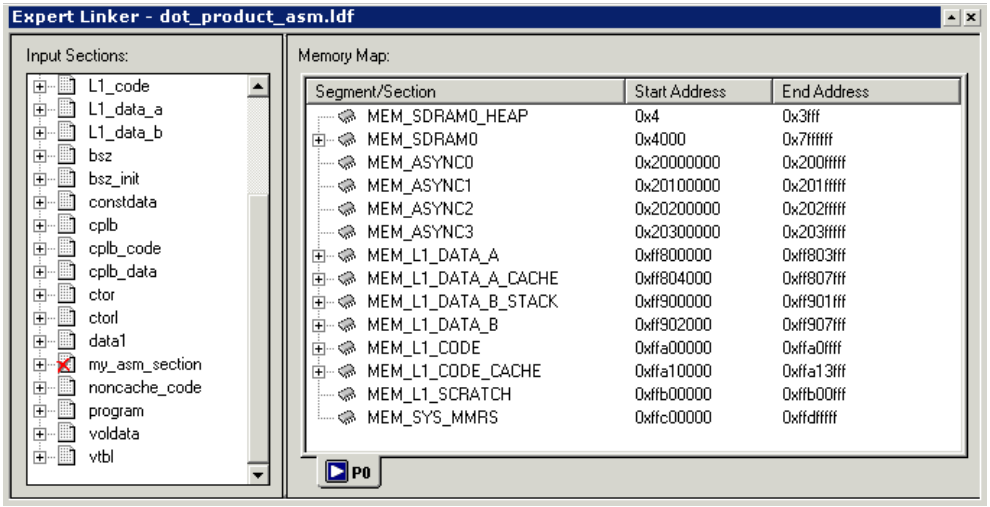

Figure 2-21. Expert Linker Window

The left pane (**Input Sections**) contains a list of the input sections that are in your project or are mapped in the `.LDF` file. A red `X` is over the icon in front of the section named "`my_asm_section`" because Expert Linker has determined that the section is not mapped by the `.LDF` file.

The right pane (**Memory Map**) contains a representation of the memory segments that Expert Linker defined when it created the `.LDF` file.

6. Map `my_asm_section` into the memory segment named `MEM_PROGRAM` as follows.

   In the **Input Sections** pane, open `my_asm_section` by clicking on the plus sign in front of it. The input section expands to show that the linker macros `$COMMAND_LINE_OBJECTS` and `$OBJECTS` and the object file `dotprod_func.doj` have a section that has not been mapped. In the **Memory Map** pane, expand `MEM_L1_CODE` and drag the icon in front of `$OBJECTS` onto the `program_ram` output section under `MEM_L1_CODE`.

   As shown in Figure 2-22, the red `X` should no longer appear because the section `my_asm_section` has been mapped.

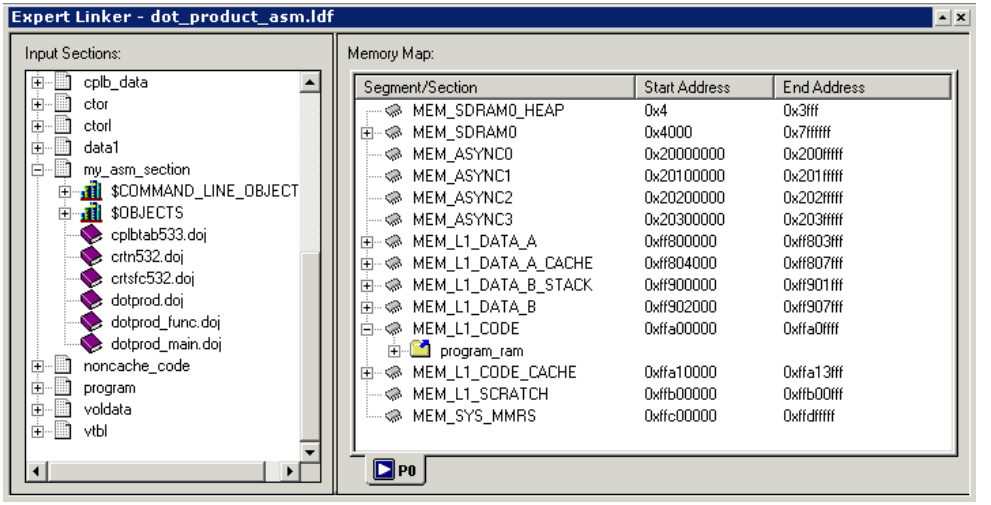

Figure 2-22. Dragging $OBJECTS onto Program Output Section

7. From the **Tools** menu, choose **Expert Linker** and **Save** to save the modified file. Then close the **Expert Linker** window.

   If you forget to save the file and then rebuild the project, VisualDSP++ will see that you modified the file and will automatically save it.

You are now ready to rebuild and run the modified project.

Step 6: Rebuild and Run dot_product_asm

To run `dot_product`:

1. Build the project by performing one of these actions:
   - Click the **Build Project** button .
   - From the **Project** menu, choose **Build Project**.

   At the end of the build, the **Output** window displays this message in the **Build** view:

   "`Build completed successfully.`"

   VisualDSP++ loads the program, runs to main, and displays the **Output**, **Disassembly**, and editor windows (shown in Figure 2-23).

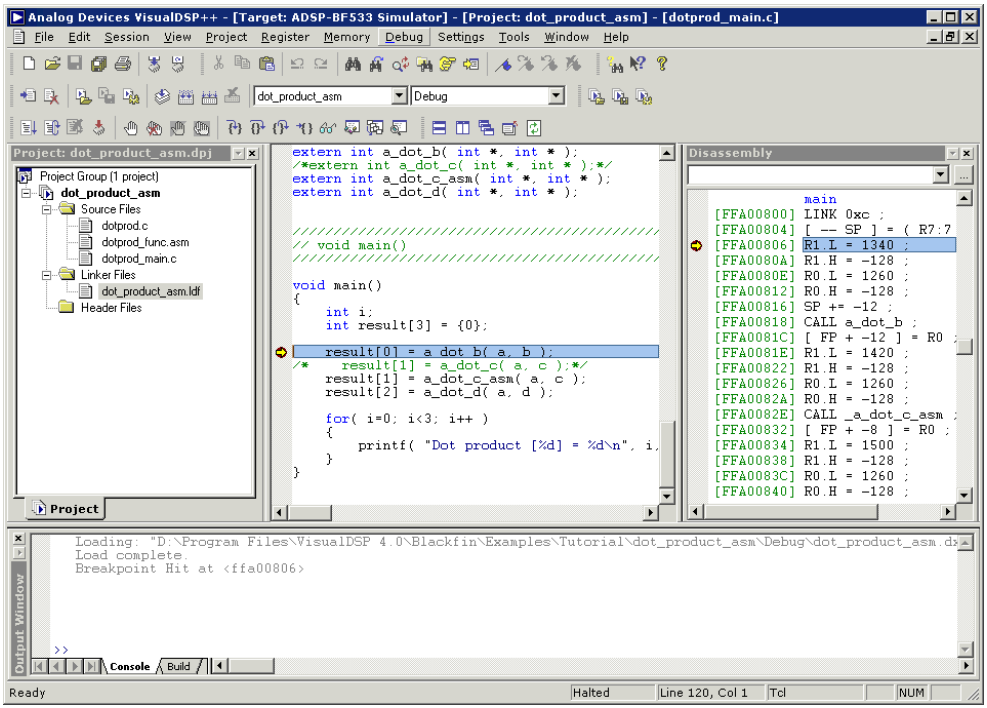
Analog Devices VisualDSP++ - [Target: ADSP-BF533 Simulator] - [Project: dot_product_asm] - [dotprod_main.c]
File Edit Session View Project Register Memory Debug Settings Tools Window Help
dot_product_asm
Debug
Project: dot_product_asm.dpj
Project Group (1 project)
dot_product_asm
Source Files
dotprod.c
dotprod_func.asm
dotprod_main.c
Linker Files
dot_product_asm.ldf
Header Files
Project
extern int a_dot_b(int *, int *);
/*extern int a_dot_c(int *, int *);*/
extern int a_dot_c_asm(int *, int *);
extern int a_dot_d(int *, int *);
// void main()
void main()
{
int i;
int result[3] = {0};
result[0] = a_dot_b(a, b);
/* result[1] = a_dot_c(a, c);*/
result[1] = a_dot_c_asm(a, c);
result[2] = a_dot_d(a, d);
for(i=0; i<3; i++)
printf("Dot product [%d] = %d\n", i,
}
Disassembly
main
[FFA00800] LINK 0xc ;
[FFA00804] [-- SP] = (R7:7
[FFA00806] R1.L = 1340 ;
[FFA0080A] R1.H = -128 ;
[FFA0080E] R0.L = 1260 ;
[FFA00812] R0.H = -128 ;
[FFA00816] SP += -12 ;
[FFA00818] CALL a_dot_b ;
[FFA0081C] [FP + -12] = R0 ;
[FFA0081E] R1.L = 1420 ;
[FFA00822] R1.H = -128 ;
[FFA00826] R0.L = 1260 ;
[FFA0082A] R0.H = -128 ;
[FFA0082E] CALL _a_dot_c_asm ;
[FFA00832] [FP + -8] = R0 ;
[FFA00834] R1.L = 1500 ;
[FFA00838] R1.H = -128 ;
[FFA0083C] R0.L = 1260 ;
[FFA00840] R0.H = -128 ;
Loading "D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\dot_product_asm\Debug\dot_product_asm.dx
Load complete.
Breakpoint Hit at <ffa00806>
Output Window
Console
Build
Ready
Halted
Line 120, Col 1
Tcl
NUM

Figure 2-23. Windows Left Open from the Previous Debugger Session

2. Click the **Run** button to run `dot_product_asm`.

   The program calculates the three dot products and displays the results in the **Console** view of the **Output** window. When the program stops running, the message "`Halted`" appears in the status bar at the bottom of the VisualDSP++ main window. The results, shown below, are identical to the results obtained in Exercise One.

```
Dot product [0] = 13273595
Dot product [1] = -49956078
Dot product [2] = 35872518
```

You are now ready to begin Exercise Three.

Exercise Three: Plotting Data

In this exercise, you:

- Load and debug a prebuilt program that applies a simple Finite Impulse Response (FIR) filter to a buffer of data
- Use VisualDSP++'s plotting engine to view the different data arrays graphically, both before and after running the program

Step 1: Load the FIR Program

To load the FIR program:

1. Keep the **Disassembly** window and **Console** page (of the **Output** window) open, but close all other windows.
2. From the **File** menu, choose **Load Program** or click . The **Open a Processor Program** dialog box appears.
3. Select the FIR program to load as follows.
    a. Open the `Analog Devices` folder and double-click:

    `VisualDSP 4.0\Blackfin\Examples\Tutorial\fir`

    b. Double-click the `Debug` subfolder.

    c. Double-click `FIR.DXE` to load the program.

    If VisualDSP++ does not open an editor window (shown in Figure 2-24), right-click in the **Disassembly** window and select **View Source**.

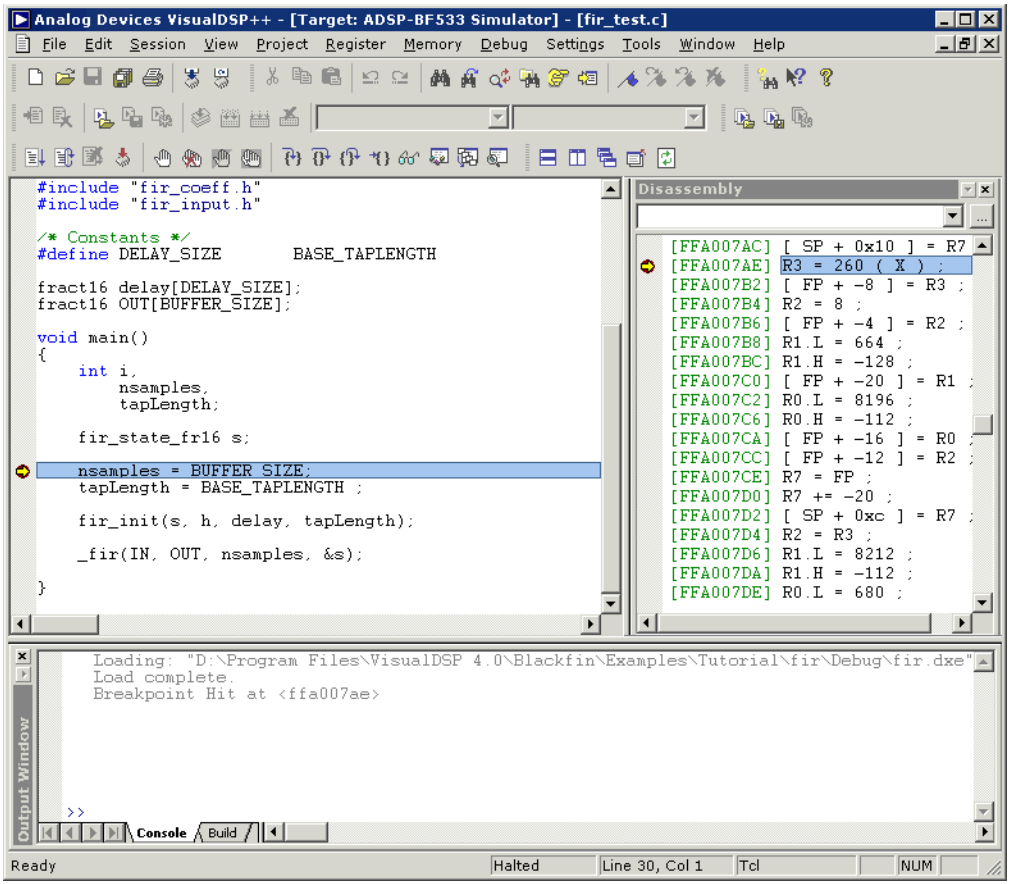
Analog Devices VisualDSP++ - [Target: ADSP-BF533 Simulator] - [fir_test.c]
File Edit Session View Project Register Memory Debug Settings Tools Window Help
#include "fir_coeff.h"
#include "fir_input.h"
/* Constants */
#define DELAY_SIZE BASE_TAPLENGTH
fract16 delay[DELAY_SIZE];
fract16 OUT[BUFFER_SIZE];
void main()
{
int i,
nsamples,
tapLength;
fir_state_fr16 s;
nsamples = BUFFER_SIZE;
tapLength = BASE_TAPLENGTH ;
fir_init(s, h, delay, tapLength);
_fir(IN, OUT, nsamples, &s);
}
Disassembly
[FFA007AC] [SP + 0x10] = R7
[FFA007AE] R3 = 260 (X) ;
[FFA007B2] [FP + -8] = R3 ;
[FFA007B4] R2 = 8 ;
[FFA007B6] [FP + -4] = R2 ;
[FFA007B8] R1.L = 664 ;
[FFA007BC] R1.H = -128 ;
[FFA007C0] [FP + -20] = R1 ;
[FFA007C2] R0.L = 8196 ;
[FFA007C6] R0.H = -112 ;
[FFA007CA] [FP + -16] = R0 ;
[FFA007CC] [FP + -12] = R2 ;
[FFA007CE] R7 = FP ;
[FFA007D0] R7 += -20 ;
[FFA007D2] [SP + 0xc] = R7 ;
[FFA007D4] R2 = R3 ;
[FFA007D6] R1.L = 8212 ;
[FFA007DA] R1.H = -112 ;
[FFA007DE] R0.L = 680 ;
Loading: "D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\fir\Debug\fir.dxe"
Load complete.
Breakpoint Hit at <ffa007ae>
Output Window
Console Build
Ready
Halted
Line 30, Col 1
Tcl
NUM

Figure 2-24. Loading the FIR Program

4. Look at the source code of the FIR program.

   You can see two global data arrays:

   - `IN`
   - `OUT`

   You can also see one function, `fir`, that operates on these arrays.

You are now ready to open a plot window.

Step 2: Open a Plot Window

To open a plot window:

1. From the **View** menu, choose **Debug Windows** and **Plot**. Then choose **New** to open the **Plot Configuration** dialog box, shown in Figure 2-25.

   Here you add the data sets that you want to view in a plot window.

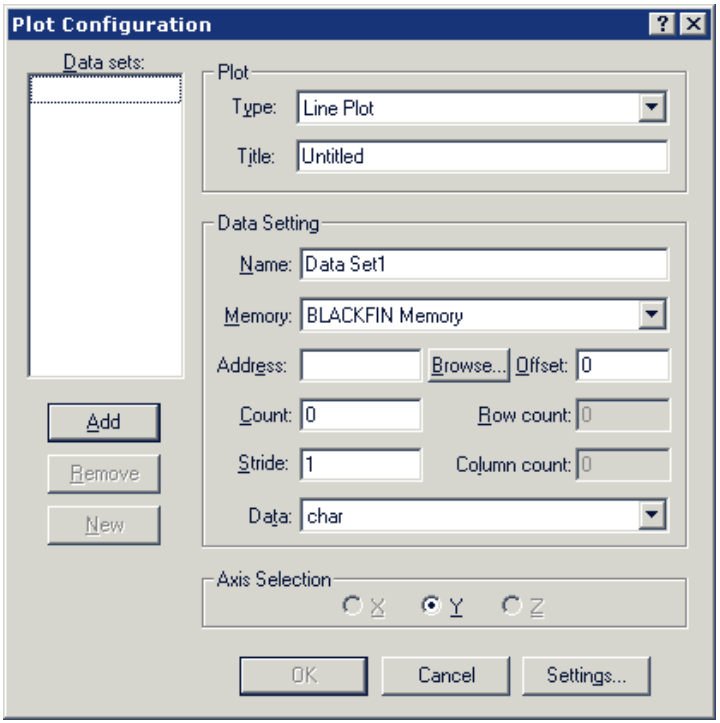

Figure 2-25. Plot Configuration Dialog Box

2. In the **Plot** group box, specify the following values.
    - In the **Type** box, select **Line Plot** from the drop-down list.
    - In the **Title** box, type `fir`.
3. Enter two data sets to plot by using the values in Table 2-4.

Table 2-4. Two Data Sets: Input and Output

Box	Input Data Set	Output Data Set	Description
Name	Input	Output	Data set
Memory	BLACKFIN Memory	BLACKFIN Memory	Data memory
Address	IN	OUT	The address of this data set is that of the Input or Output array. Click **Browse** to select the value from the list of loaded symbols.
Count	128	128	The array is 260 elements long, but you are plotting the first 128 elements.
Stride	1	1	The data is contiguous in memory.
Data	short	short	Input and Output are arrays of int values.

After entering each data set, click **Add** to add the data set to the **Data sets** list on the left of the dialog box.

The **Plot Configuration** dialog box should now look like the one in Figure 2-26.

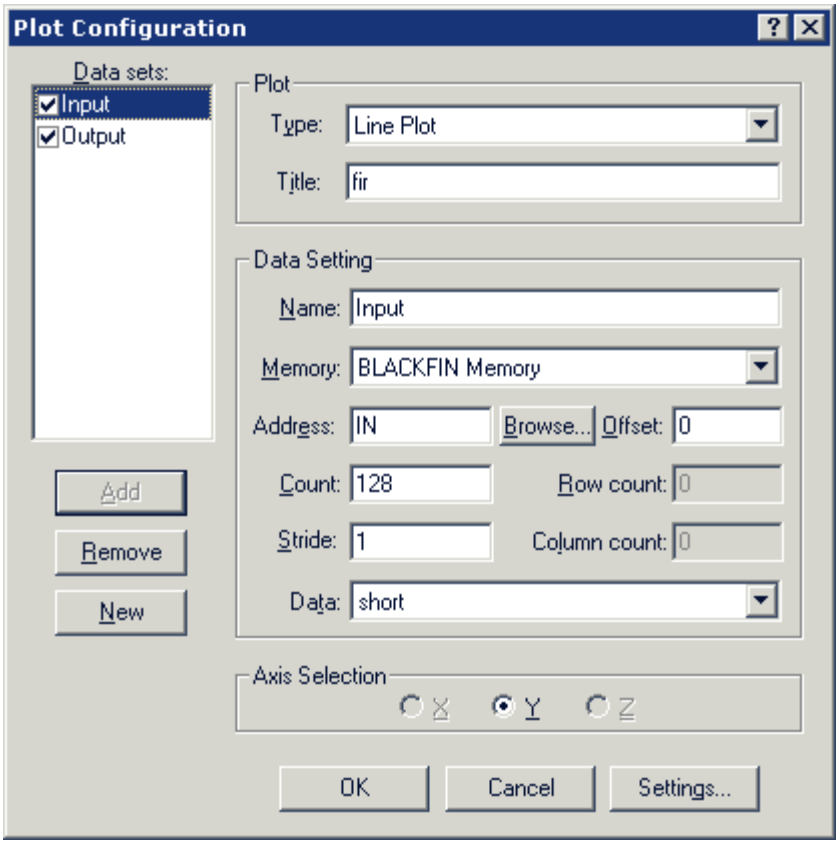

Figure 2-26. Plot Configuration Dialog Box with Input/Output Data Sets

4. Click **OK** to apply the changes and to open a plot window with these data sets.

   The plot window now displays the two arrays. By default, the simulator initializes memory to zero, so the **Output** data set appears as one horizontal line, shown in Figure 2-27.

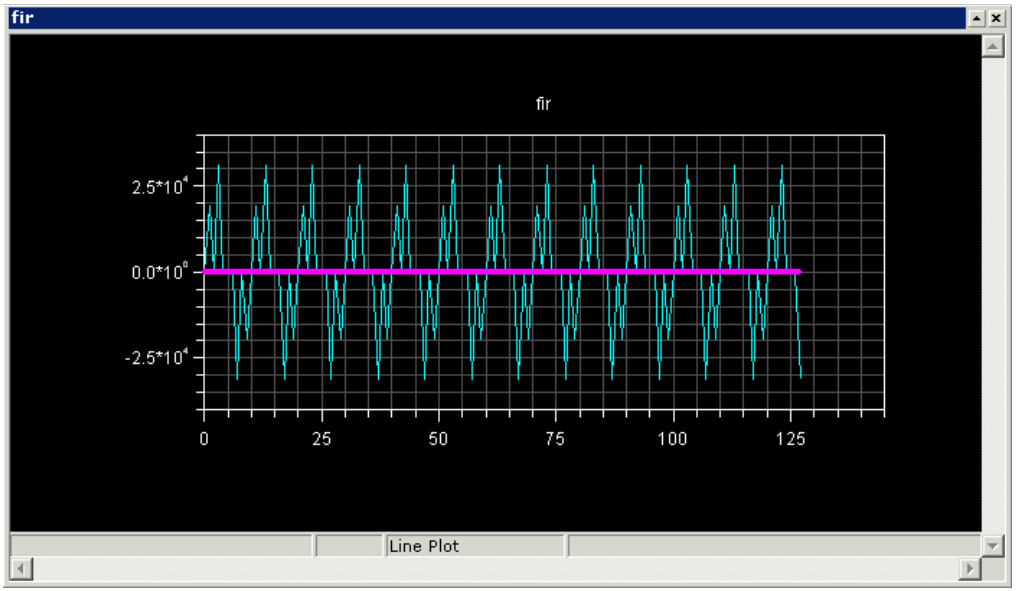

Figure 2-27. Plot Window: Before Running the FIR Program

Resizing the plot window changes the scale on the x and y axis.

5. Right-click in the plot window and choose **Modify Settings**. On the **General** page, in the **Options** group box, select **Legend** and click **OK** to display the legend box.

Step 3: Run the FIR Program and View the Data

To run the FIR program and view the data:

1. Press **F5** or click the **Run** button to run to the end of the program.

   When the program halts, you see the results of the FIR filter in the Output array. The two data sets are visible in the plot window, as shown in Figure 2-28.

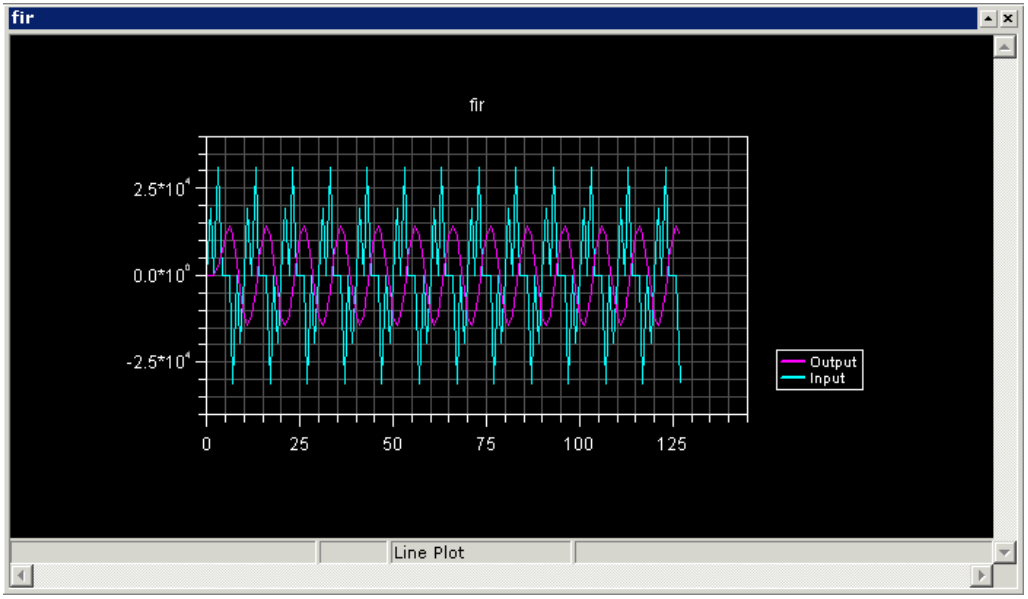

Figure 2-28. Plot Window After Running the FIR Program to Completion

Next you will zoom in on a particular region of interest in the plot window to focus in on the data.

2. Click the left mouse button inside the plot window and drag the mouse to create a rectangle around the area of interest. Then release the mouse button to magnify the selected region.

   Figure 2-29 shows the selected region, and Figure 2-30 shows the magnified result.

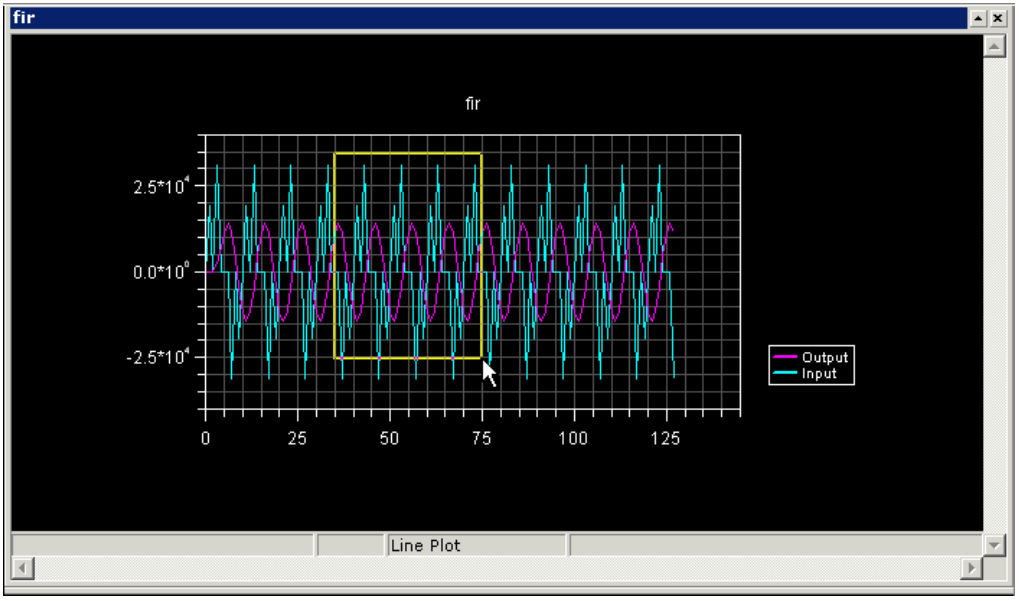

Figure 2-29. Plot Window: Selecting a Region to Magnify

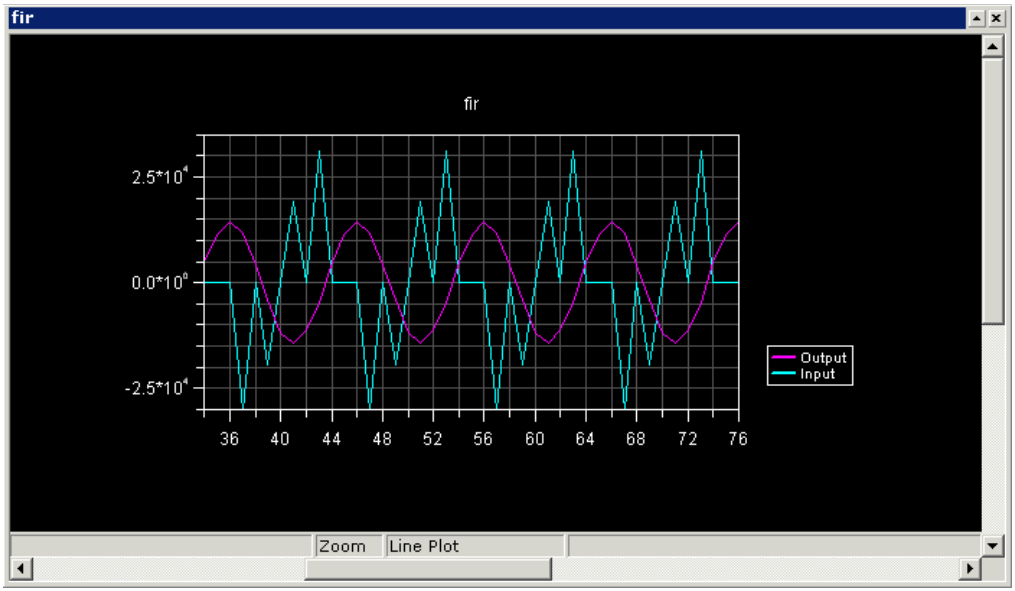

Figure 2-30. Plot Window: Magnified Result

To return to the previous view (before magnification), right-click in the plot window and choose **Reset Zoom** from the pop-up menu. You can view individual data points in the plot window by enabling the data cursor, as explained in the next step.

3. Right-click inside the plot window and choose **Data Cursor** from the pop-up menu. Move to each data point in the current data set by pressing and holding the Left (←) or Right (→) arrow key on the keyboard. To switch data sets, press the Up (↑) or Down (↓) arrow key. The value of the current data point appears in the lower-left corner of the plot window, as shown in Figure 2-31.

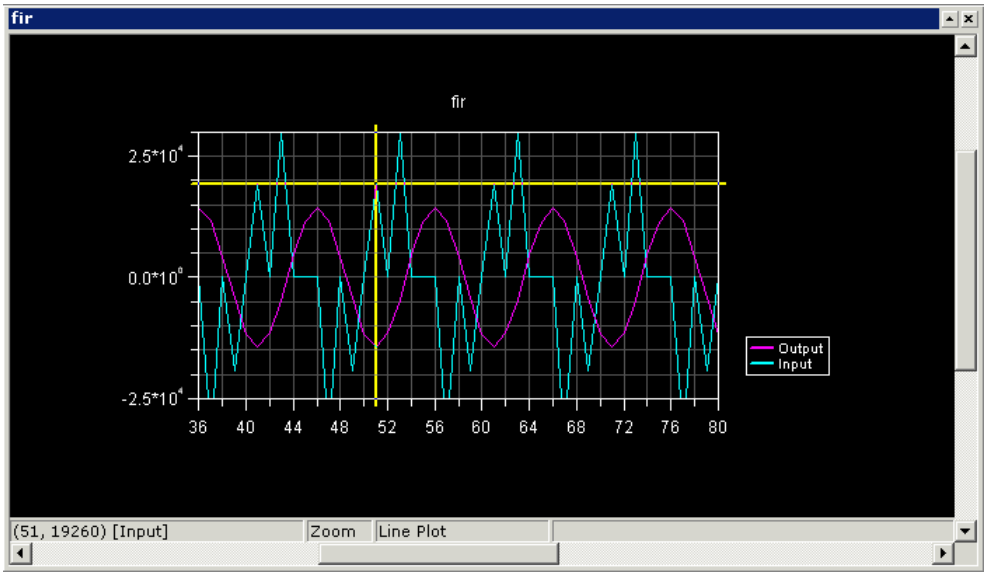

Figure 2-31. Plot Window: Using the Data Cursor Feature

4. Right-click in the plot window and choose **Data Cursor** from the pop-up menu.

   Next you will look at data sets in the frequency domain.

5. Right-click in the plot window and choose **Modify Settings** to open the **Plot Settings** dialog box.

6. Complete these steps:

    a. Click the **Data Processing** tab to display the **Data Processing** page, shown in Figure 2-32.

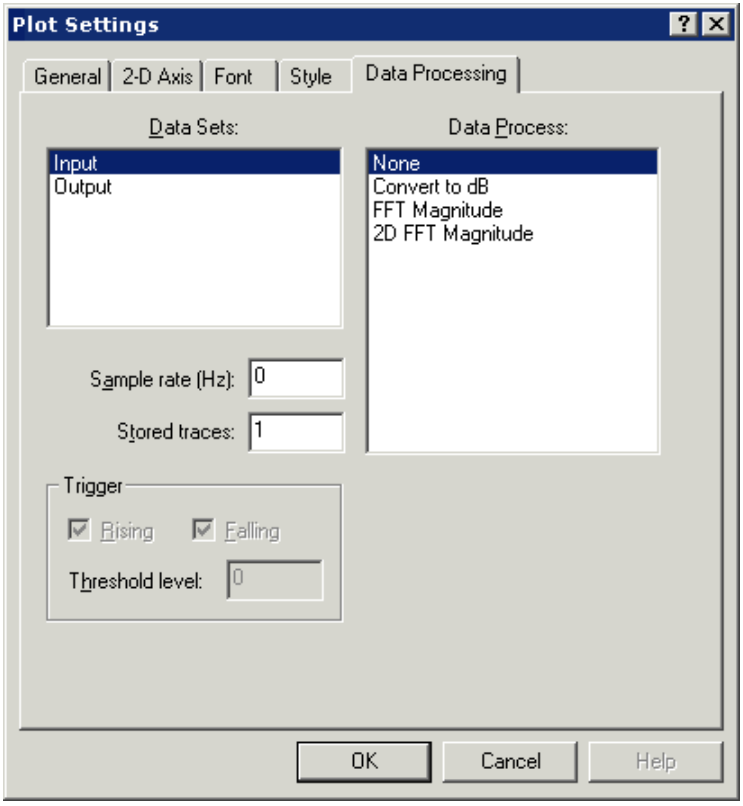

Figure 2-32. Data Processing Page

    b. In the **Data Sets** box, ensure that **Input** (the default) is selected. In the **Data Process** box, choose **FFT Magnitude**.
    c. In the **Sample rate (Hz)** box, type **10000**.
    d. In the **Data Sets** box, select **Output**. In the **Data Process** box, choose **FFT Magnitude**

e. Click **OK** to exit the **Plot Settings** dialog box.

VisualDSP++ performs a Fast Fourier Transform (FFT) on the selected data set before it is plotted. The FFT enables you to view the signal in the frequency domain, as shown in Figure 2-33.

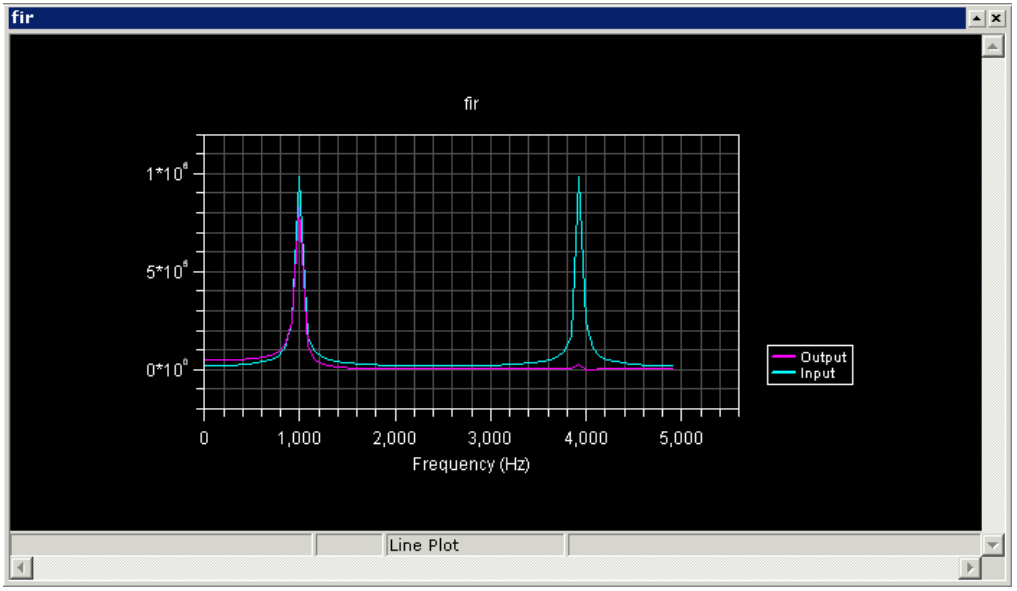

Figure 2-33. FFT Performed on a Selected Data Set

Now, complete the following steps to look at the FIR filter's response in the frequency domain.

1. From the **View** menu, choose **Debug Windows** and **Plot**. Then choose **New** to open the **Plot Configuration** dialog box.
2. Set up the Filter Frequency Response plot by completing the **Plot** and **Data Setting** group boxes as shown in Figure 2-34.

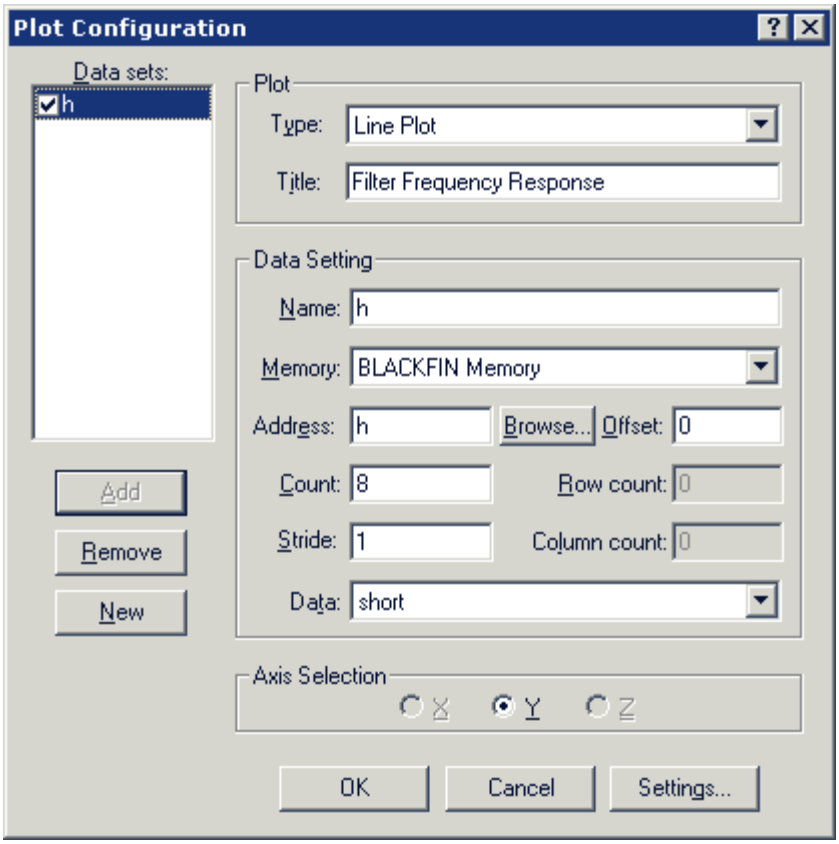

Figure 2-34. Filter Frequency Response Data Set

3. Click **Add** to add the data set to the **Data sets** box.
4. Click **OK** to apply the changes and to open the plot window with this data set.
5. Right-click in the plot window and choose **Modify Settings** to open the **Plot Settings** dialog box.

6. Click the **Data Processing** tab to display the **Data Processing** page, shown in Figure 2-32 on page 2-47. Complete this page as follows.

    a. In the **Data Sets** box, select **h**.

    b. In the **Data Process** box, choose **FFT Magnitude**.

    c. In the **Sample rate (Hz)** box, type **10000**.

    d. Click **OK** to exit the **Data Processing** page.

    VisualDSP++ performs a Fast Fourier Transform (FFT) on the selected data set, and enables you to view the filter response plot in the frequency domain, as shown in Figure 2-35.

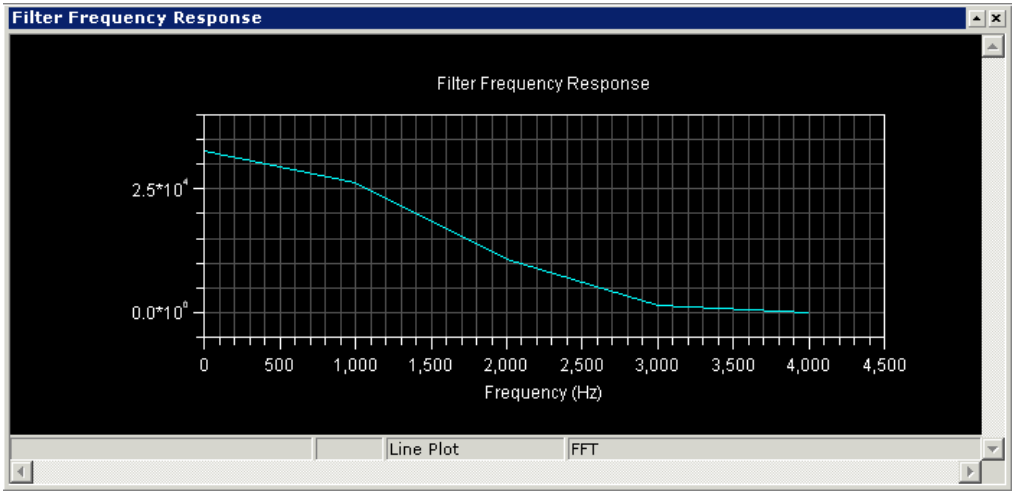

Figure 2-35. Filter Frequency Response Plot

This plot shows that the low-pass FIR filter cuts off all frequency components above 4,000 Hz. When you apply a low-pass filter to the input signal, the resulting signal has no output above 4,000 Hz.

You are now ready to begin Exercise Four.

Exercise Four: Linear Profiling

In this exercise, you:

- Load and debug the FIR program from the previous exercise
- Use linear profiling to evaluate the program's efficiency and to determine where the application is spending the majority of its execution time in the code

VisualDSP++ supports two types of profiling: linear and statistical.

- You use linear profiling with a simulator. The count in the **Linear Profiling Results** window is incremented every time an assembly instruction is executed.
- You use statistical profiling with a JTAG emulator connected to a processor target. The count in the **Statistical Profiling Results** window is based on random sampling of the program counter.

Step 1: Load the FIR Program

To load the FIR program:

1. Close all open windows except the **Disassembly** window and the **Output** window.
2. From the **File** menu, choose **Load Program**, or click . The **Open a Processor Program** dialog box appears.

3. Select the program to load as follows.
    a. Open the `Analog Devices` folder and double-click:

       `VisualDSP 4.0\Blackfin\Examples\Tutorial\fir`

    b. Double-click the `Debug` subfolder.
    c. Double-click `FIR.DXE` to load and run the FIR program.

       If VisualDSP++ does not open an editor window (shown in Figure 2-37), right-click in the **Disassembly** window and select **View Source**.

You are now ready to set up linear profiling.

Step 2: Open the Profiling Window

To open the **Linear Profiling Results** window:

1. From the **Tools** menu, choose **Linear Profiling** and then choose **New Profile**.

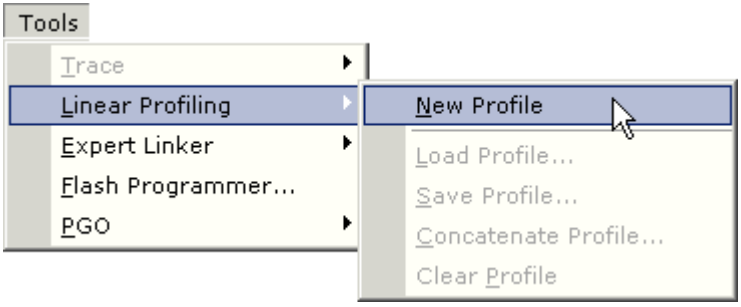

Figure 2-36. Setting Up Linear Profiling for the FIR Program

The **Linear Profiling Results** window opens without any data.

2. Click in the profiling window's title bar and then drag and drop the window to the top of the VisualDSP++ main window, as shown in Figure 2-37. You will have a better view of the profile data.

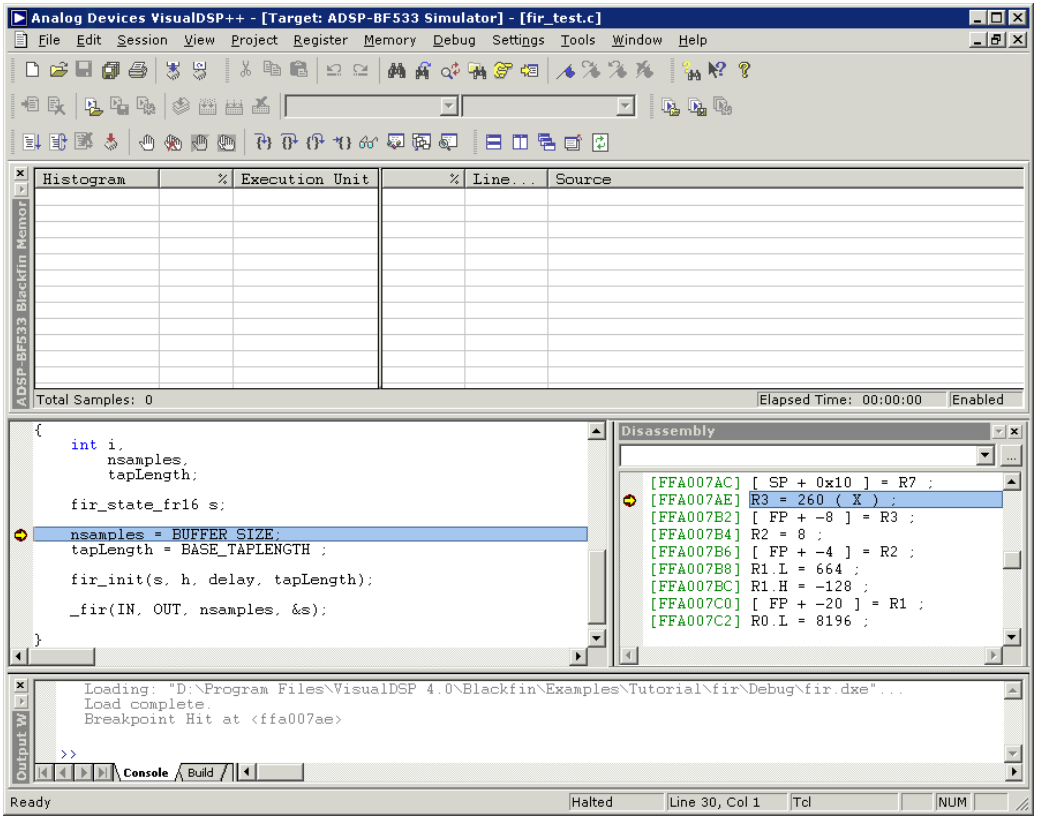
Analog Devices VisualDSP++ - [Target: ADSP-BF533 Simulator] - [fir_test.c]
File Edit Session View Project Register Memory Debug Settings Tools Window Help
Histogram % Execution Unit % Line... Source
ADSP-BF533 Blackfin Memor
Total Samples: 0
Elapsed Time: 00:00:00
Enabled
{
int i,
nsamples,
tapLength;
fir_state_fr16 s;
nsamples = BUFFER_SIZE;
tapLength = BASE_TAPLENGTH ;
fir_init(s, h, delay, tapLength);
_fir(IN, OUT, nsamples, &s);
}
Disassembly
[FFA007AC] [SP + 0x10] = R7 ;
[FFA007AE] R3 = 260 (X) ;
[FFA007B2] [FP + -8] = R3 ;
[FFA007B4] R2 = 8 ;
[FFA007B6] [FP + -4] = R2 ;
[FFA007B8] R1.L = 664 ;
[FFA007BC] R1.H = -128 ;
[FFA007C0] [FP + -20] = R1 ;
[FFA007C2] R0.L = 8196 ;
Loading: "D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\fir\Debug\fir.dxe"...
Load complete.
Breakpoint Hit at <ffa007ae>
Output W
Console Build
Ready
Halted
Line 30, Col 1
Tcl
NUM

Figure 2-37. Linear Profiling Results Window (Empty)

The **Linear Profiling Results** window is initially empty. Linear profiling is performed when you run the FIR program. After you run the program and collect data, this window displays the results of the profiling session.

You are now ready to collect and examine linear profile data.

Step 3: Collect and Examine the Linear Profile Data

To collect and examine the linear profile data:

1. Press **F5** or click to run to the end of the program.

   When the program halts, the results of the linear profile appear in the **Linear Profiling Results** window.

2. Examine the results of your linear profiling session.

   The **Linear Profiling Results** window is divided into two three-column panes.

   The left pane shows the results of the profile data. You can see the percentages of total execution time consumed, by function and by address.

   Double-clicking a line with a function enables you to display the source file that contains that function. For example, double-click the `fir` function to display the assembly source file (`fir.asm`) in the right pane, as shown in Figure 2-38.

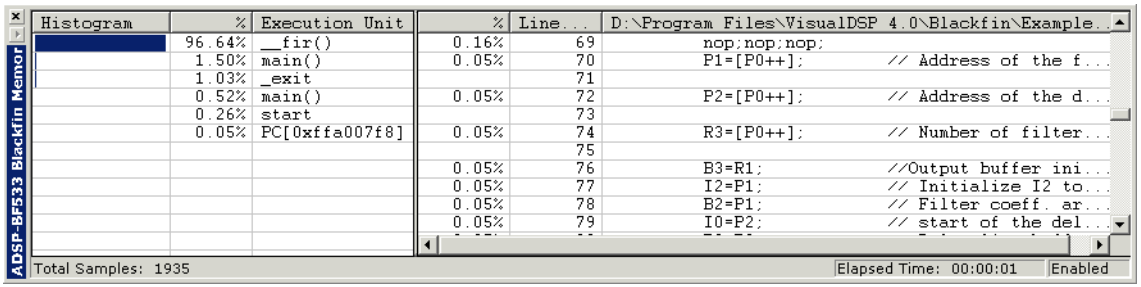

Figure 2-38. Linear Profiling Results, FIR Program Performance Analysis

The field values in the left pane are defined as follows.

Histogram	A graphical representation of the percentage of time spent in a particular execution unit. This percentage is based on the total time that the program spent running, so longer bars denote more time spent in a particular execution unit. The **Linear Profiling Results** window always sorts the data with the most time-consuming (expensive) execution units at the top.
%	The numerical percent of the same data found in the Histogram column. You can view this value as an absolute number of samples by right-clicking in the **Linear Profiling Results** window and by selecting **View Sample Count** from the pop-up menu.
Execution Unit	The program location to which the samples belong. If the instructions are inside a C function or a C++ method, the execution unit is the name of the function or method. For instructions that have no corresponding symbolic names, such as hand-coded assembly or source files compiled without debugging information, this value is an address in the form of `PC[xxx]`, where `xxx` is the address of the instruction. If the instructions are part of an assembly file, the execution unit is either an assembly function or the assembly file followed by the line number in parentheses.

In Figure 2-38 on page 2-54 the left pane shows that the `fir` function consumes over 93% of the total execution time. The right (source) pane, shown in Figure 2-39, displays the percentage that each line in the `fir` function consumes.

%	Line...	D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tu...
	66	__fir :
	67	
0.05%	68	P0=[SP+12]; // Address of the filt...
0.16%	69	nop;nop;nop;
0.05%	70	P1=[P0++]; // Address of the filte...
	71	
0.05%	72	P2=[P0++]; // Address of the delay...
	73	
0.05%	74	R3=[P0++]; // Number of filter coe...
	75	
0.05%	76	B3=R1; //Output buffer initial...
0.05%	77	I2=P1; // Initialize I2 to the...
0.05%	78	B2=P1; // Filter coeff. array ...
0.05%	79	I0=P2; // start of the delay l...
0.05%	80	B0=P2; // Delay line buffer is...
0.05%	81	I1=P2; // start of the delay l...
0.05%	82	B1=P2; // Delay line buffer is...
	83	
0.05%	84	I3=R1;
0.05%	85	P1=R2;
0.05%	86	P2=R3;
	87	
0.05%	88	R2=R2+R2;
0.05%	89	CC=BITTST(R3,0); //Check if the number o...
0.05%	90	R3=R3+R3; //As the filter coeff. ...
0.05%	91	L2=R3; //Initialize the filter...
0.05%	92	P0=R0; // Address of the inpu...
	93	
0.26%	94	IF !CC JUMP FIR_CONTINUE (BP);
	95	R3+=2; //Make the filter taps ...
	96	L2=R3;
	97	NOP;NOP;NOP;NOP;
	98	I2-=2; // Location where zer...
	99	R0=0;
	100	W[I2++]=R0.L; //Set the last filter ...
	101	//force the number of fi...

Figure 2-39. Linear Profile Data for fir.asm

You have now completed the Basic Tutorial.

3 ADVANCED TUTORIAL

This chapter contains the following topics.

- "Overview" on page 3-1
- "Exercise One: Using Profile-Guided Optimization" on page 3-2
- "Exercise Two: Using Background Telemetry Channel" on page 3-22

Overview

This tutorial demonstrates advanced features and techniques that you can use in the VisualDSP++ Integrated Development and Debugging Environment (IDDE). The exercises use sample programs written in C and assembly for Blackfin processors.

- In **Exercise One: Using Profile-Guided Optimization**, you build a project with PGO support, create PGO files, compile the project without using the information in the PGO files, recompile the project by using the PGO files to optimize the build, check the PGO results, and compare execution times.
- In **Exercise Two: Using Background Telemetry Channel**, you add BTC to your DSP application and then run two demos that demonstrate BTC functionality.

The ADSP-BF53x Family Simulator and ADSP-BF533 processor are used in Exercise One. The EZ-KIT Lite USB emulator, HPPCI-ICE, and ADSP-BF533 processor are used in Exercise Two.

Exercise One: Using Profile-Guided Optimization

Profile-guided optimization (PGO) is an optimization technique that uses collected profile information to guide the compiler optimizer's decisions.

Traditionally, a compiler compiles each function only once and attempts to produce generated code that will perform well in most cases. The compiler has to make decisions about the best code to generate. For example, given an `if…then…else` construct, the compiler has to decide whether the most common case is the `then` or the `else`. You can offer crude guidelines—compile for speed or compile for space—but, usually, the compiler has to make a default decision.

With PGO, the compiler makes these decisions based on data collected during previous executions of the generated code. This process involves the following steps.

1. Compiling the application to collect profile information
2. Running the application in a simulator session by using representative data sets

   The simulator accumulates profile data indicating where the application spends most of its time.
3. Recompiling the application by using the collected profile data

   The compiler uses the collected information rather than the application's default behavior to make decisions about the relative importance of parts of the application.

The profile data collected from a simulator run is stored in a file with a `.PGO` suffix. You can process multiple data sets to cover the spectrum of potential data and create a separate `.PGO` file for each data set. The recompilation stage can accept multiple `.PGO` files as input.

You must complete these basic steps to use PGO:

1. Build the application with PGO support.
2. Set up one or more streams in the simulator to provide a set of data inputs that represent what the application would see in a real target environment.
3. Tell the simulator to produce a `.PGO` file with a specified file name.
4. Load and run the application to produce the `.PGO` file.
5. Rebuild the application and pass all `.PGO` files to the compiler, which uses the generated PGO results to optimize the application.

In this exercise, you:

- Load the PGO example project in the VisualDSP++ environment
- Create data sets for profile-guided optimization
- Attach input streams to the data sets
- Create `.PGO` files by executing the project with the data sets as input
- Recompile the project by using the `.PGO` files to optimize the build
- Run the optimized version of the project with the same data sets as input
- Compare the execution times of all three executions

The files used for this exercise are in the `pgo` folder. The default installation path of this folder is:

```
Program Files\Analog Devices\VisualDSP++ 4.0\Blackfin\Examples\
Tutorial\pgo
```

Step 1: Load the Project

To open a VisualDSP++ project:

1. Start VisualDSP++ and connect to an ADSP-BF533 simulator session. For information about connecting to a session, refer to "Step 1: Start VisualDSP++ and Open a Project" on page 2-4.

2. Open the `PgoExample.dpj` project. For details about opening projects, see "Step 1: Start VisualDSP++ and Open a Project" on page 2-4.

   This project contains a C file, `PgoExample.c`. When you run the program, it reads data from an address and counts the number of even and odd values. This counting is done with an `if…then…else` statement. If the majority of values read are odd, the program will spend most of its time in the `then…` branch. If the majority of values are even, the program will spend most of its time in the `else…` branch. Normally, the compiler has no way of knowing which branch will be taken more often. By using PGO, the compiler can determine which branch is used most often and optimize the next build.

This project also contains a Visual Basic script that demonstrates how to use the VisualDSP++ Automation API to perform PGO. The automation functionality is beyond the scope of this tutorial. Refer to online Help for more information about automation.

Three data files are used as input to the C program. These simple text files contain lists of values.

- `Dataset_1.dat` has 128 even values (50%) and 128 odd values (50%).
- `Dataset_2.dat` has 192 even values (75%) and 64 odd values (25%).
- `Dataset_3.dat` has 256 even values (100%) and 0 odd values (0%).

To view these files, choose the **Open** command on the **File** menu in VisualDSP++. The two possible values in all three files are either `0x01` or `0x02`. Each file contains 256 values.

In this exercise, assume that this program will be used in the real world, and that you can expect a similar distribution of values as input from the real world.

By looking at the C code and the potential input, you can easily see that the executed program will spend more time in the `else…` branch than in the `then…` branch. Without using PGO, the compiler cannot make this same conclusion. By default, it will expect the `then…` branch to be executed most frequently and will compile the code without optimizing execution time.

Since the example program and input are very simple, you could fix the problem by making a few minor changes to the code. Manually tweaking a large program to speed up execution time, however, would take far too long, and you would have to analyze sample input on your own. PGO provides a quick and easy way to enable the compiler to make these adjustments for you.

Step 2: Configure a Data Set

The first step in the PGO process is to create a *data set*—a collection of sample input for the program being optimized. A data set feeds the input into the executing program, and this input causes the program to be executed along certain paths. Some paths will be used more often than others. This information is recorded by the simulator and stored in a `.PGO` file for the compiler to use later for optimization. The most commonly used paths will be optimized to run quickly, and less common paths will run more slowly.

To create the first of three data sets for this exercise:

1. From the **Tools** menu, choose **PGO** and then **Manage Data Sets**, as shown in Figure 3-1.

Figure 3-1. Manage Data Sets Menu Option

The **Manage Data Sets** dialog box (Figure 3-2) is displayed.

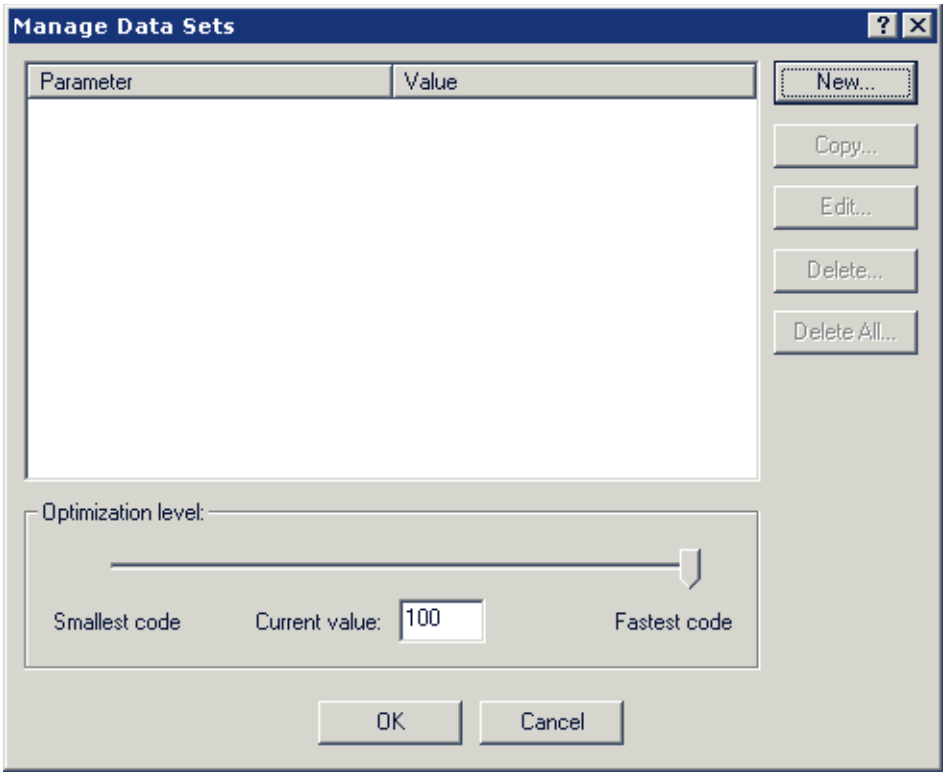

Figure 3-2. Manage Data Sets Dialog Box

This dialog box is where you manage data sets. Note the **Optimization level** slider bar. This control allows you to customize your optimization. Moving the slider all the way to the left enables you to build as small an executable as possible, but may sacrifice execution speed. Moving the slider all the way to the right enables you to build a fast executable, with a potential space tradeoff. Placing the slider between the two extremes provides varying ratios of space versus speed optimization. For this exercise, move the slider all the way to the right.

2. Click the **New** button to open the **Edit Data Set** dialog box, shown in Figure 3-3.

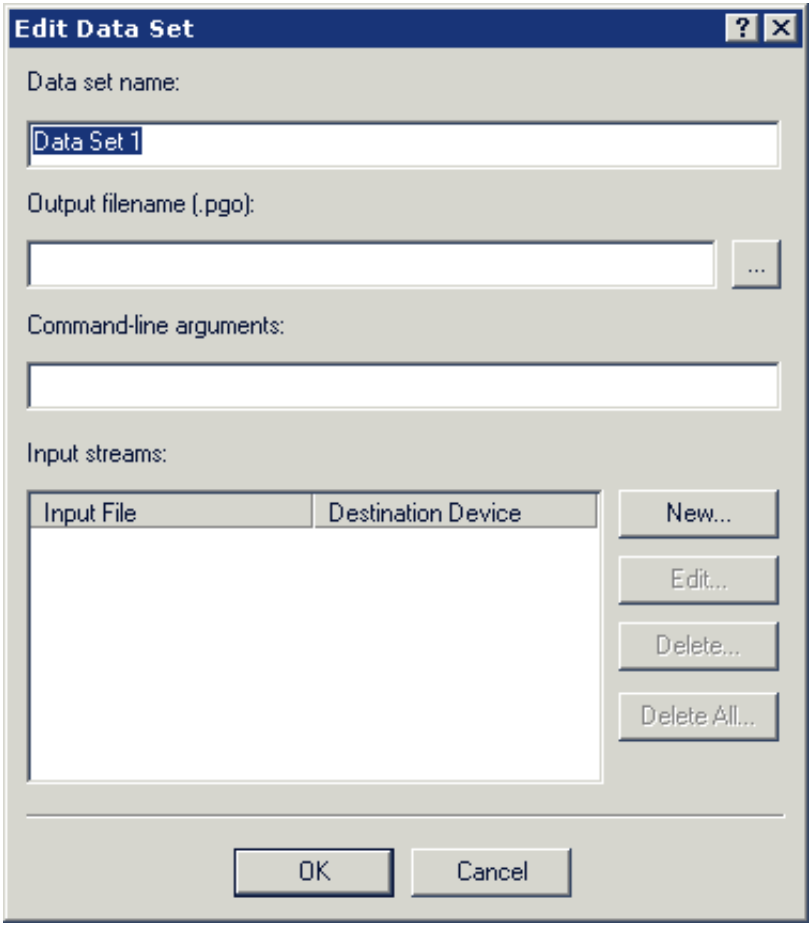

Figure 3-3. Edit Data Set Dialog Box

3. Replace the default **Data set name** with a more descriptive name. Since the first data file contains an equal number of even and odd values, use a name such as `50% Even - 50% Odd`.

4. Specify the **Output filename** (where the optimization information produced by this data set will be saved). Optimization information is saved in files with a `.PGO` suffix.

   Type in a file name such as `dataset_1.pgo`. The file will be saved in the project directory. To save the files elsewhere, type in a full path name. You can use **Command line arguments** for more advanced control of the data set, but they are not covered in this tutorial.

   For more information about command-line arguments, see the *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for Blackfin Processors*.

Now you have to attach an input stream to this data set.

Step 3: Attach an Input Stream

In this step you attach an input stream to the data set.

1. Click the **New** button to open the **Edit PGO Stream** dialog box, shown in Figure 3-4.

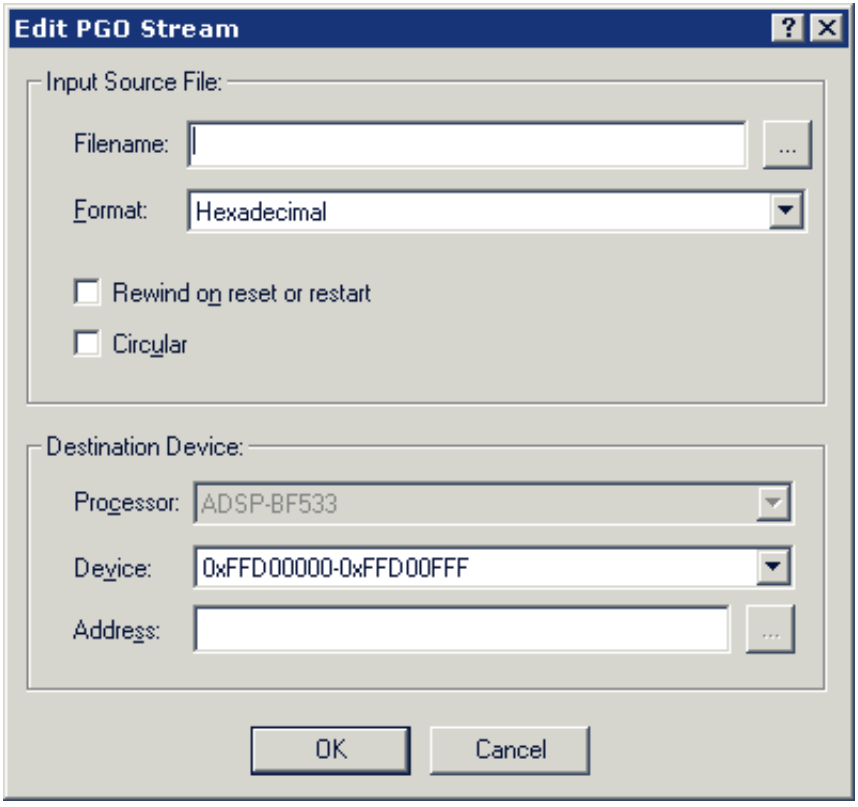

Figure 3-4. Edit PGO Stream Dialog Box

An input stream maps a data file to a destination device. In this exercise, the input streams map the three data files to the simulator. The input stream provides the program with input as needed during execution.

For more information about streams, see the "Debugging" chapter in the *VisualDSP++ 4.0 User's Guide*.

2. Complete the **Input Source File** group box as described in Table 3-1.

Table 3-1. Input Source File Group Box Settings

Field/Control	Action/Value
Filename	Specify a file name by clicking the file browse button and selecting the input source file `dataset_1.dat` from the `pgo` directory.
Format	The data in this file is in hexadecimal format, so leave the format setting as is.
Rewind on reset or restart	Select this option. When you run a program with an input stream, the program may or may not work through all of the data in the stream. If the program encounters a reset or restart event before working through the entire data stream and this option is enabled, the next execution starts at the beginning of the input stream. Otherwise, execution continues where it left off.
Circular	Select this option. It allows the program to read through an input stream many times during a single execution.

3. In the **Destination Device** group box, specify where the data from the input stream is sent. Refer to Table 3-2.

Table 3-2. Destination Device Group Box Settings

Field/Control	Action/Value
Processor	This field lets you specify a peripheral in another processor as the destination device. For this tutorial, you are connected to a single processor session, so this field is disabled.
Device	This field lets you choose any stream device supported by the simulator target as the destination. Devices can include a memory address or various peripherals. Available devices depend on the processor you are using. For more information on devices, see the hardware manual for your processor. The program reads the input streams from memory, so leave this field as it is.
Address	Specify where in memory the input will be sent. Since the program in this exercise reads data from address `0xFFD00000` (refer to `PgoExample.c`), enter this value.

The completed dialog box should now look like Figure 3-5.

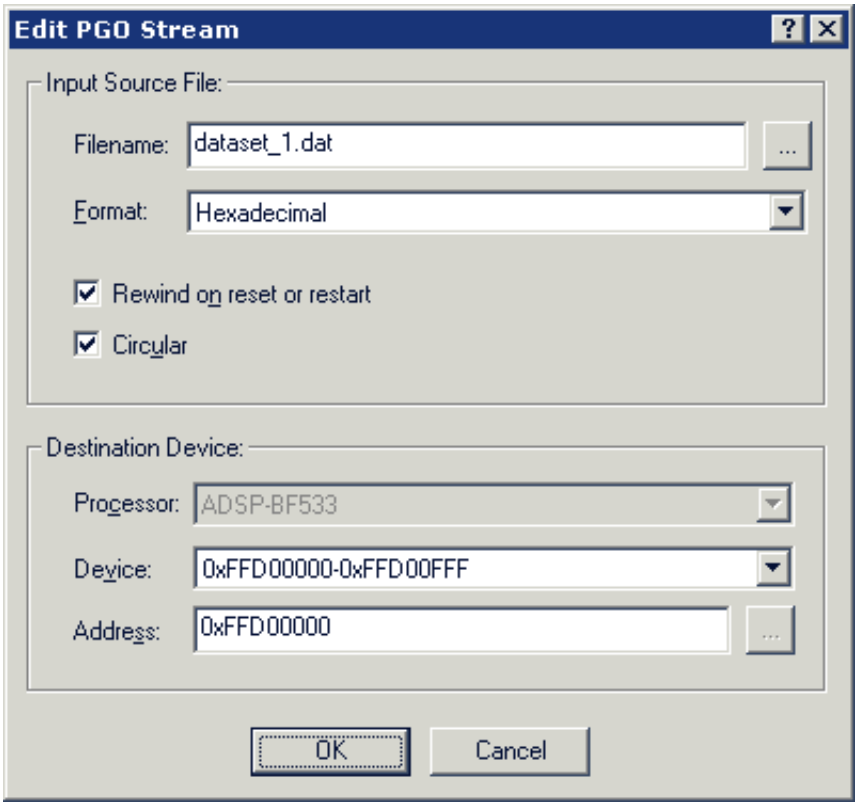

Figure 3-5. A Configured PGO Stream

4. Click **OK** to return to the **Edit Data Set** dialog box. The dialog box with your configured data set should match Figure 3-6.

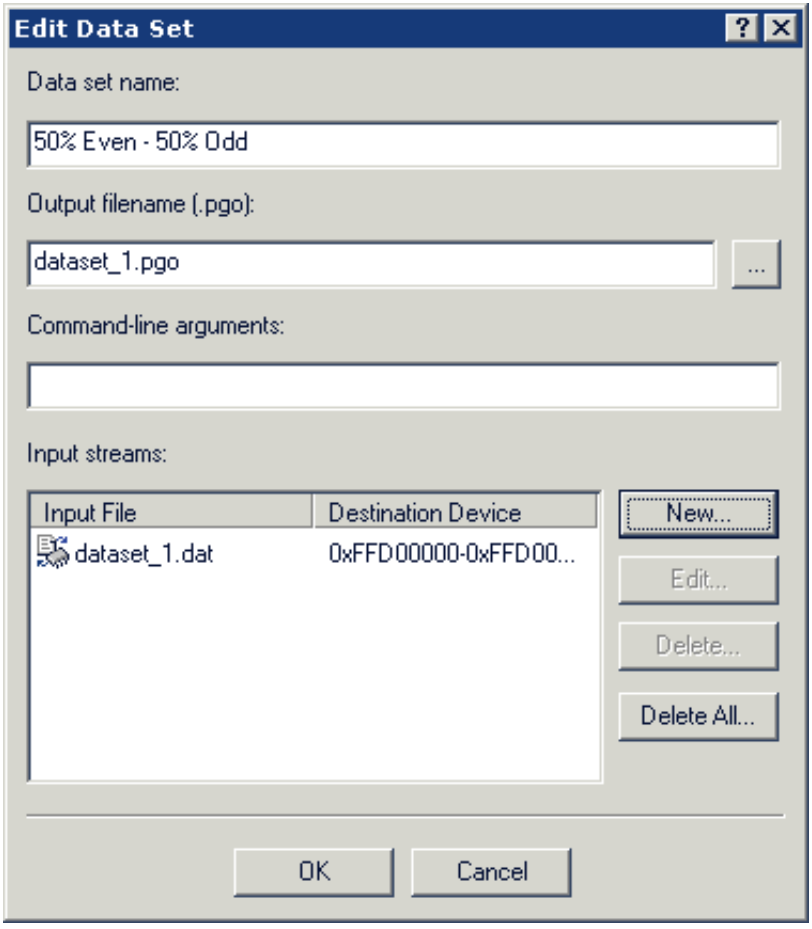

Figure 3-6. A Configured Data Set

5. Click **OK** to save the data set and close the dialog box.

You now have to create the remaining two data sets.

Step 4: Configure Additional Data Sets

To create the remaining two data sets, you can repeat the steps used to create the first data set and substitute the appropriate files, or use the **Copy** button.

The following steps explain how to use the **Copy** button to create a data set.

1. Highlight the `50% Even - 50% Odd` data set, and click the **Copy** button.

   The **Edit Data Set** dialog box opens with the information for the `50% Even - 50% Odd` data set. Clicking the **OK** button makes a copy of the `50% Even - 50% Odd` data set. For this exercise, however, you will edit the data set.

2. In the **Data set name** field, specify an appropriate name for the new data set.

   The second input source file contains three times as many even values as odd values, so use a name such as `75% Even - 25% Odd`.

3. In the **Output filename** field, type the name `dataset_2.pgo` to save the `.PGO` file in the project directory.

   To save the file elsewhere, click the file browse button (...) to specify a full path.

4. In the **Input streams** box, highlight the `dataset_1.dat` **Input File** and click the **Edit** button.

5. Click the file browse button (...) to change the **Input Source File** from `dataset_1.dat` to `dataset_2.dat`.

6. Click the **OK** button to return to the **Edit Data Set** dialog box.

   The second data set is now complete.

7. Click the **OK** button to return to the **Manage Data Sets** dialog box.

8. Create the third data set from scratch or modify a copy of one of the existing data sets.

   Make sure that you use the `dataset_3.pgo` and `dataset_3.dat` files. The third data set contains all even values, so give it a name such as `100% Even - 0% Odd`. When you are finished, expand the three data sets listed in the **Manage Data Sets** dialog box and compare them with the data sets in Figure 3-7.

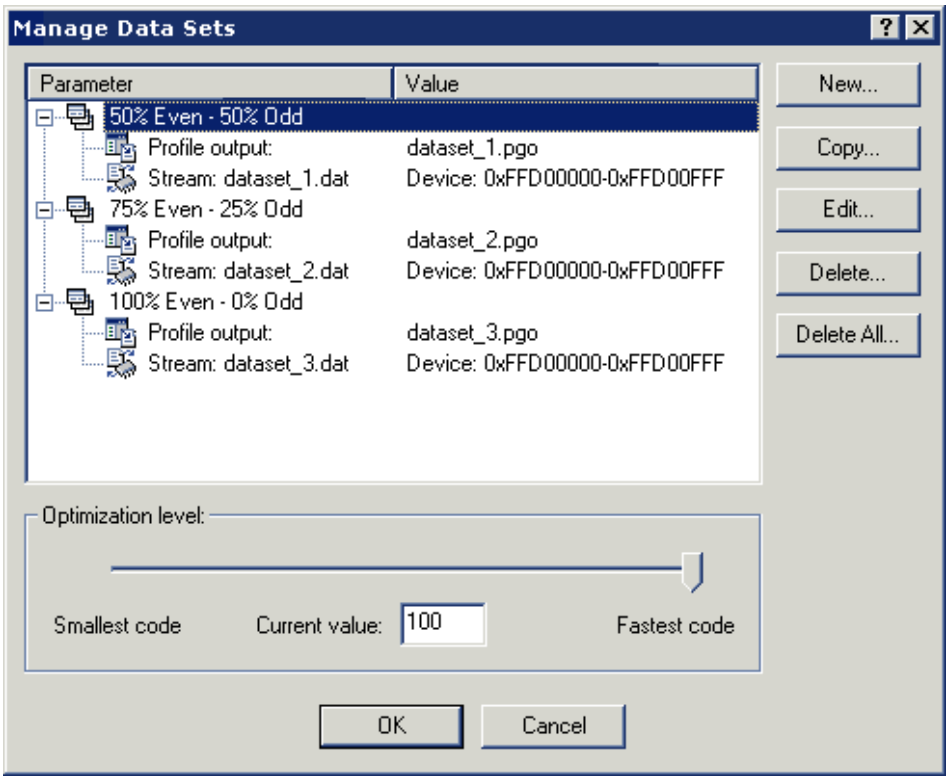

Figure 3-7. Expanded Data Sets

If your data sets match those in Figure 3-7, you have the data sets needed to optimize the program.

9. Click **OK** to save the data sets and close the dialog box.

You are now ready to create `.PGO` files.

Step 5: Create PGO Files and Optimize the Program

Now that you have configured the data sets, you are ready to optimize your program.

From the **Tools** menu, choose **PGO** and then **Execute Data Sets**, as shown in Figure 3-8.

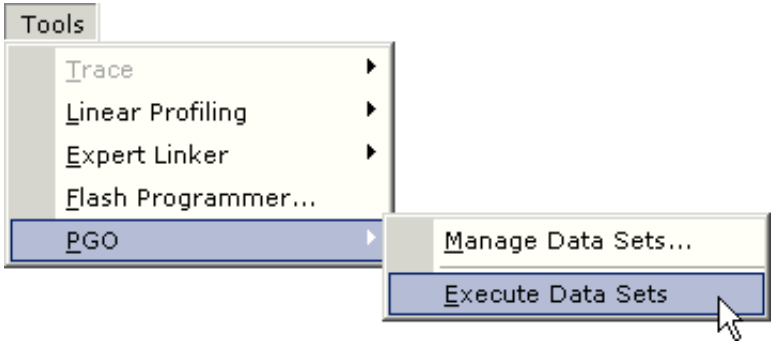

Figure 3-8. Execute Data Sets Menu Option

Several things happen during the execute process. First, the project is built with the `-pguide` switch, which enables the collection of the PGO data that is later fed back into the compiler. The compiler makes default assumptions about which sections of code will be most commonly executed. Next, the resulting executable is run once with each data set. While the program is running, the simulator monitors the paths of execution through the program, and the number of cycles used in the execution. As stated before, this information is stored in the `.PGO` file that you specified when creating each data set.

Once the program has been run with each data set, the project is recompiled. This time, however, the compiler uses the information found in the `.PGO` files to optimize the resulting executable. This optimized executable is then run with the input provided by each data set, and again, the simulator monitors each execution.

You are now ready to examine the results of the optimization.

Step 6: Compare Execution Times

When the execution is completed, an XML report of the PGO optimization results is generated and displayed in a browser window. This file is in the `pgo\debug` folder and is named `PgoReport.`*`date and time`*`.xml` (for example, `PgoReport.20031027145428.xml`).

At the top of the report is a header, shown in Figure 3-9.

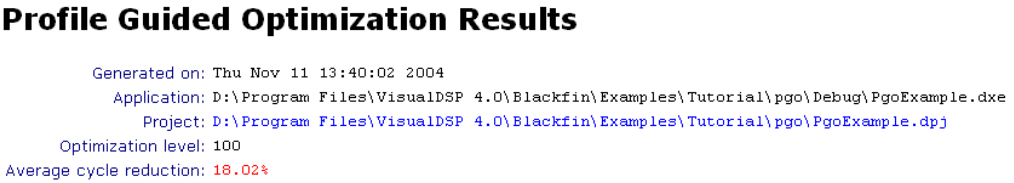
Profile Guided Optimization Results
Generated on: Thu Nov 11 13:40:02 2004
Application: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\Debug\PgoExample.dxe
Project: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\PgoExample.dpj
Optimization level: 100
Average cycle reduction: 18.02%

Figure 3-9. PGO Results – Report Header

The header provides basic information such as the project name, location, and when the report was generated. Also listed is the optimization level (which you specified with the slider bar in the **Manage Data Sets** dialog box, Figure 3-2 on page 3-7), and an average result. The **Average result** is the difference in total cycle counts on all executions from before and after optimization.

The **Average result** obtained on your machine may vary slightly from the result shown in Figure 3-9.

The header is followed by information about each data set (see Figure 3-10).

Data Set: 50% Even - 50% Odd
Command line:
Input stream: File: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\dataset_1.dat
Device: 0xFFD00000-0xFFD00FFF
PGO output: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\dataset_1.pgo
Before optimization: 7875 cycles
After optimization: 7875 cycles
Cycle reduction: 0.00%
Data Set: 75% Even - 25% Odd
Command line:
Input stream: File: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\dataset_2.dat
Device: 0xFFD00000-0xFFD00FFF
PGO output: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\dataset_2.pgo
Before optimization: 8713 cycles
After optimization: 7043 cycles
Cycle reduction: 19.17%
Data Set: 100% Even - 0% Odd
Command line:
Input stream: File: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\dataset_3.dat
Device: 0xFFD00000-0xFFD00FFF
PGO output: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\dataset_3.pgo
Before optimization: 9539 cycles
After optimization: 6211 cycles
Cycle reduction: 34.89%

Figure 3-10. PGO Results – Data Sets

The file information, including the **Data Set** file name, **Input stream** file name, and **PGO output** file name, is listed first. Then the results of optimization are shown. The number of cycles needed to run the original build with this data set (**Before optimization**) is followed by the number of cycles needed to run this data set on the optimized build (**After optimization**). Note that the number of cycles may vary on different machines.

Finally, the percent difference between the two builds (**Result**) is listed. A positive percentage indicates that the optimized build ran faster than the original build.

The **Execution Output** section of the log appears first. Figure 3-11 shows selections from the execution output.

Execution Output

```
------------------------------------------------------------------------
 Executing PGO Data Sets
------------------------------------------------------------------------

Building application with PGO support...
        Build complete.

Profiling Data Set: "50% Even - 50% Odd"...
        Loading application: PgoExample.dxe
        Setting command line:
        Creating input stream: File: dataset_1.dat -> Device: 0xFFD00000-0xFFD00FFF
        Setting PGO output: dataset_1.pgo
        Running application: PgoExample.dxe
Breakpoint Hit at <ffa0074e>

        Profile results: 7875 cycles
```

Figure 3-11. PGO Results – Execution Output Sample

This information is the output that appeared in the **Console** view of the **Output** window while the PGO was running. The output includes the basic events that occurred during execution.

The **Build Output** section appears next at the bottom of the report. This section contains build output for each build. Figure 3-12 shows a build output sample.

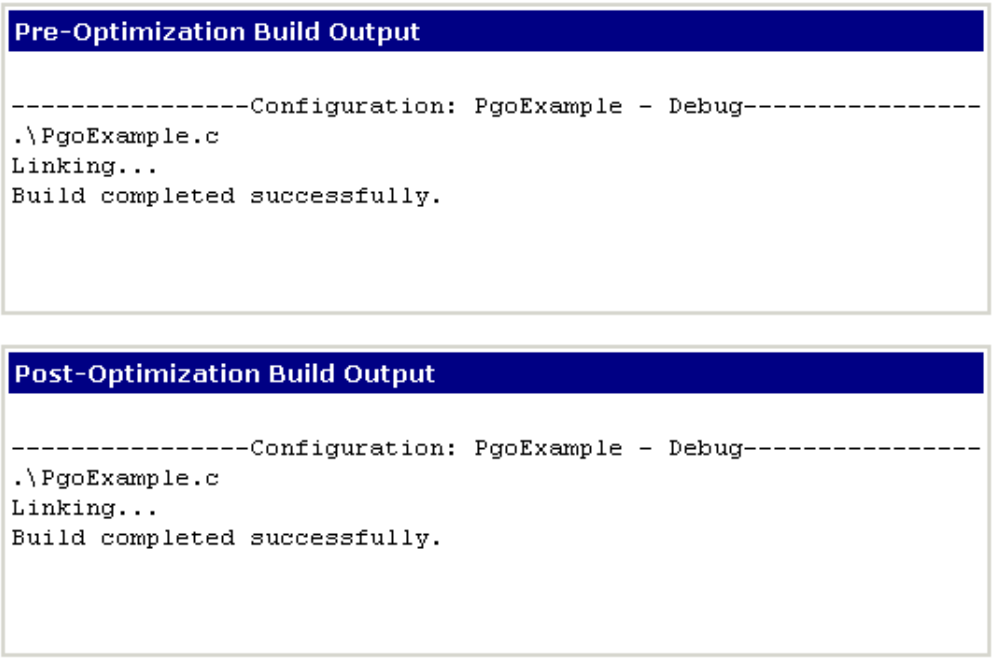

Figure 3-12. PGO Results – Build Output Sample

This information is the output that was displayed in the **Build** view of the **Output** window while the PGO was running.

This output information shows how effective PGO can be. As shown in Figure 3-9 on page 3-18, the optimized executions used approximately 18% fewer cycles than the original executions. The gain in performance is significant, especially given the ease with which it was accomplished.

You are now ready to begin Exercise Two.

Exercise Two: Using Background Telemetry Channel

A background telemetry channel (BTC) enables you to exchange data between a host and target application without halting the processor. This mechanism provides real-time visibility into a running program. Uses for a BTC include:

- Monitoring program status without halting the processor
- Viewing algorithm output in real time
- Injecting data into a program
- Streaming real-time data to a file (data logging)
- Providing I/O, either standard or user-defined

In this exercise, you:

- Add BTC to your DSP application.
- Run the BTC Assembly demo, designed to demonstrate the basic functionality of BTC.
- Run the BTC FFT demo, which demonstrates the transfer of data from the Blackfin EZ-KIT Lite over background telemetry channels.

Adding BTC to Your DSP Application

To add BTC to your DSP application:

1. Add the `btc.h` header file to your source code.

   The `btc.h` file contains the macros necessary to use BTC in your programs, including the macros necessary to define a channel.

2. Define one or more channels in the DSP source code.

   Channels identify an address range for data transfer. Each channel consists of a name, a start address, and a length. All channels are defined inside a single BTC map block. A sample BTC channel definition is shown below:

   ```
   #include “btc.h”
   short AudioData[12000]//audio capture array
   BTC_MAP_BEGIN
   BTC_MAP_ENTRY( “Audio Data Channel”, (long)&AudioData,
   sizeof(AudioData))
   BTC_MAP_END
   ```

3. Define the BTC polling loop with the `btc_poll()` command.

   The polling loop checks for incoming commands from the host. The location of the polling loop determines the polling priority, which affects application performance. You can place the polling loop in a high-priority interrupt, a low-priority interrupt, or in the main program loop.

4. Initialize BTC by using the `btc_init()` command.

5. Add the BTC library, `libbtc532.dlb` or `libbtc535.dlb`, to the project.

   The BTC library contains the functions that transfer data to and from the host.

Several example programs included with VisualDSP++ 4.0 can help you become familiar with BTC. The following demos walk you step-by-step through two of the Blackfin ADSP-BF533 examples. Analog Devices highly recommends connecting to an ADSP-BF533 EZ-KIT Lite board via HPPCI-ICE or HPUSB-ICE when running these examples.

The low data rate required enables you to run the first example by using the EZ-KIT Lite USB interface. The second example, however, requires a faster data rate than the EZ-KIT Lite USB interface can provide. Refer to the manual included with your HPPCI-ICE, HPUSB-ICE, or EZ-KIT Lite if you are unsure of how to establish a connection.

BTC is not currently supported by simulation targets.

The default installation path for these examples is:

```
Program Files\Analog Devices\VisualDSP 4.0\BlackFin\Examples\BTC
Examples\BF533
```

You are now ready to run the first BTC demo.

Running the BTC Assembly Demo

The BTC assembly demo is designed to demonstrate the basic functionality of BTC. The program defines several BTCs to allow the transfer of data over the BTC interface while the processor is running. For example, one channel counts the number of interrupts that have occurred, and another counts the number of times a push button is pressed. See the `Btc_AsmDemo.asm` header for more details. You will use the **BTC Memory** window in the IDDE to view the data in each channel.

Figure 3-13 provides an overview of data transfer over the BTC interface in the BTC assembly demo.

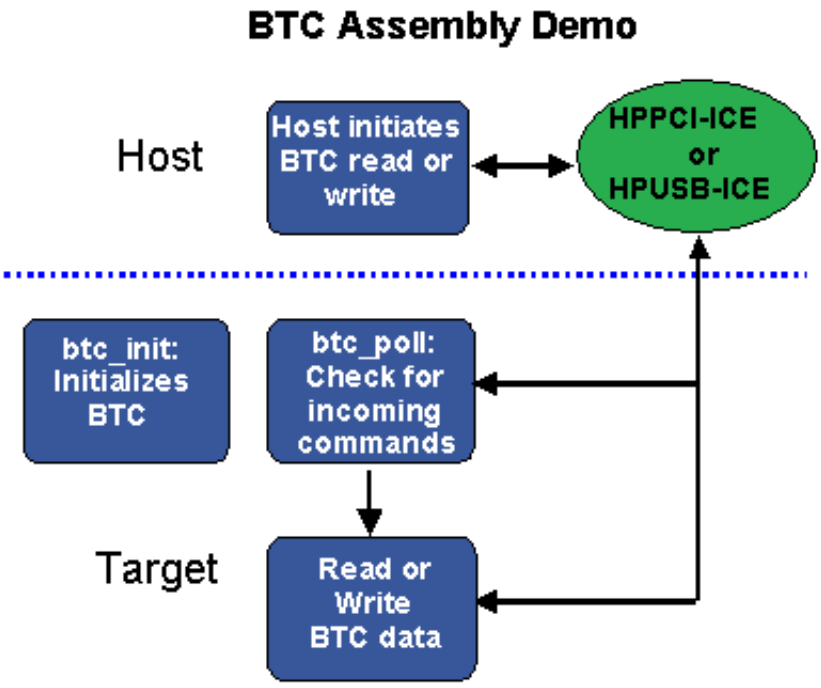

Figure 3-13. Data Transfer in the Assembly Demo

Step 1: Load the BTC_AsmDemo Project

1. Start VisualDSP++ and connect to an ADSP-BF533 HPPCI-ICE, HPUSB-ICE, or EZ-KIT Lite USB emulator session.

   For information about connecting to a session, refer to "Step 1: Start VisualDSP++ and Open a Project" on page 2-4 in Exercise One.

2. Open the `Btc_AsmDemo.dpj` project, under `Analog Devices` in:

   `VisualDSP 4.0\BlackFin\Examples\BTC Examples\BF533`

   For details about loading a project, see "Step 1: Start VisualDSP++ and Open a Project" on page 2-4.

You are now ready to examine BTC commands.

Step 2: Examine the BTC Commands

1. Open the `Btc_AsmDemo.asm` file by double-clicking on it in the **Project** window.
2. Scroll down to the section labeled `BTC Definitions` in the comments (see Figure 3-14). Notice that the seven channels are defined as described in step 2 of "Adding BTC to Your DSP Application" on page 3-22.

```
////////////////////
// BTC Definitions
////////////////////
BTC_MAP_BEGIN
//              Channel Name,                Starting Address, Length
BTC_MAP_ENTRY('Timer Interrupt Counter', timerCounter,     0x00004)
BTC_MAP_ENTRY('PF10 Counter',            pf10Counter,      0x00004)
BTC_MAP_ENTRY('PF11 Counter',            pf11Counter,      0x00004)
BTC_MAP_ENTRY('256 byte channel (PF11)', pf11Array,        0x00100)
BTC_MAP_ENTRY('4k byte channel',         array1,           0x01000)
BTC_MAP_END
```

Figure 3-14. BTC Channel Definitions

3. Scroll down to the `main program`.

   Twenty-four lines below the `_main:` label is the command to initialize BTC, `call _btc_init;` (shown in Figure 3-15).

```
//////////////////////
// main program
//////////////////////
.section program;
.extern ldf_stack_end;
.global _main;
_main:
        // initialize the stack pointer
        sp.h = ldf_stack_end;
        sp.l = ldf_stack_end;

        // setup the EVT
        [--sp] = rets;
        call initVectorRegs;
        rets = [sp++];
        // init LEDs
        [--sp] = rets;
        call initLEDs;
        rets = [sp++];
        // setup the Core Timer
        [--sp] = rets;
        call initCoreTimer;
        rets = [sp++];
        // setup the Programmable Flags
        [--sp] = rets;
        call initProgFlags;
        rets = [sp++];

        // initialize the BTC
        [--sp] = rets;
        call _btc_init;
        rets = [sp++];
```

Figure 3-15. BTC Initialize Command

For more information about `_btc_init`, refer to the VisualDSP++ Help.

In this example, the `call _btc_poll` command is placed in the `EVT15_LOOP`, defined below `main` and shown in Figure 3-16.

```
EVT15_LOOP:
        nop;
        nop;
        nop;
        [--sp] = rets;
        call  btc_poll;
        rets = [sp++];
        nop;
        nop;
        nop;
        jump EVT15_LOOP;
_main.end:
```

Figure 3-16. BTC Polling Loop

Refer to the online Help for more information about `_btc_poll`. This function is called when the `evt15` interrupt is triggered. This interrupt has the lowest priority on this particular processor.

Now that you have seen how BTC has been added to this example, it is time to build the project.

4. On the toolbar, click **Rebuild All** ( ) or choose **Rebuild All** from the **Project** menu.

   This command builds the project and automatically downloads the application to the target. For details about building projects, refer to "Exercise One: Building and Running a C Program" on page 2-3.

Step 3: Set Up the BTC Memory Window and View Data

1. From the **View** menu, choose **Debug Windows** and **BTC Memory** as shown in Figure 3-17.

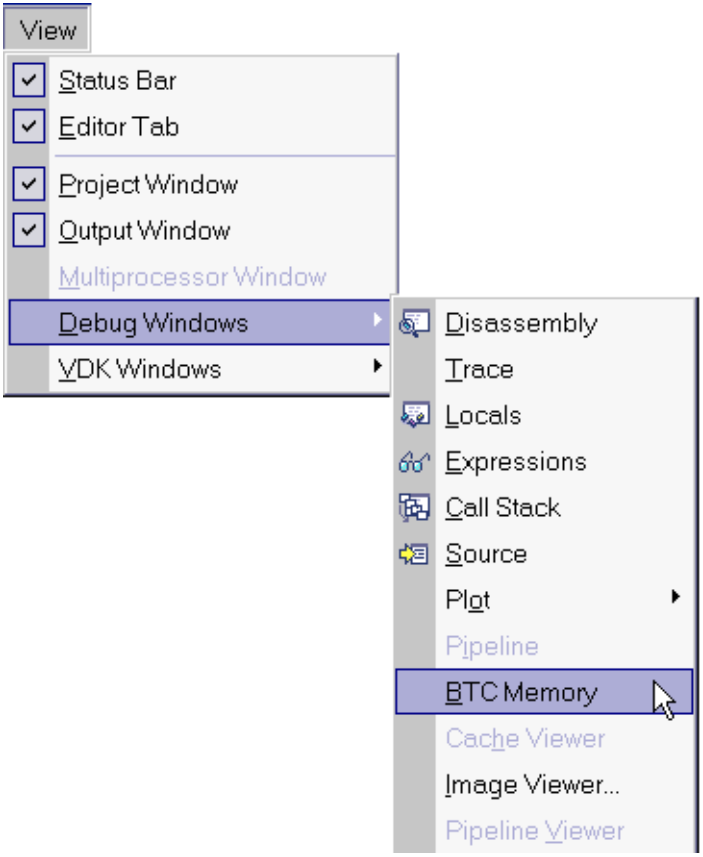

Figure 3-17. BTC Memory Menu Option

The **BTC Memory** window, shown in Figure 3-18, is displayed.

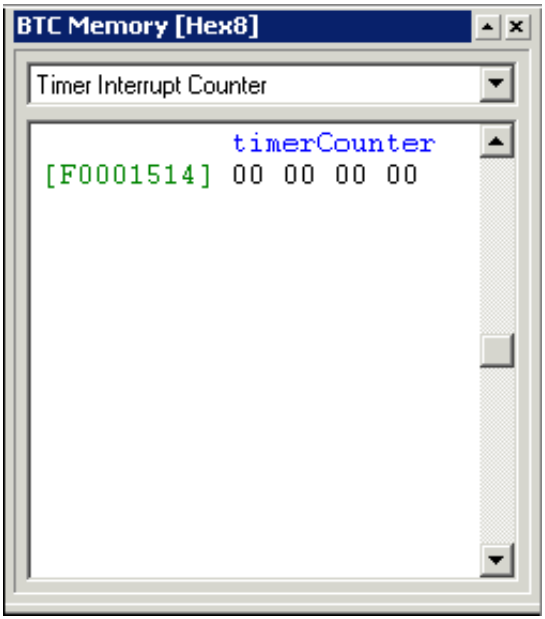

Figure 3-18. BTC Memory Window

The **BTC Memory** window displays BTC data in real time. Data is read from the target when the IDDE issues a read request, and is written when a value is edited in the **BTC Memory** window. You can adjust the rate at which the IDDE requests data by changing the refresh rate.

2. Right-click in the **BTC Memory** window to display a menu of features, shown in Figure 3-19.

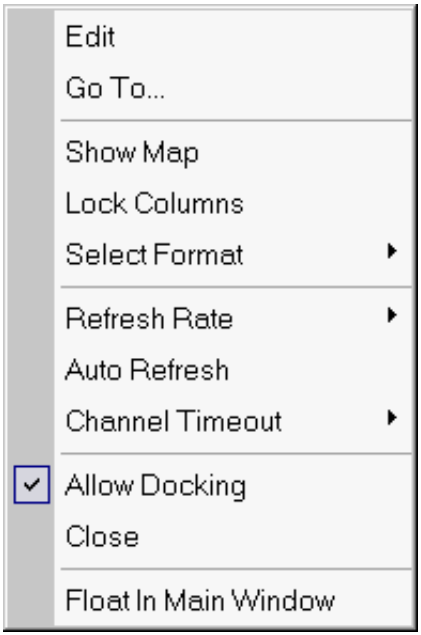

Figure 3-19. BTC Memory Window Right-Click Menu

Each menu option is described as follows.

Edit – Right-click on a memory location in the **BTC Memory** window and select **Edit** to modify the value in that location. You can also edit by left-clicking on a memory value in the window and typing in a new value.

Go To – Enables you to enter an address or browse for a symbol, and displays memory starting at that address in the **BTC Memory** window. If the address entered is outside the range of the defined BTC channels, an error message is displayed.

Show Map – When this option is enabled, a map of all the defined channels is displayed, as shown in Figure 3-20.

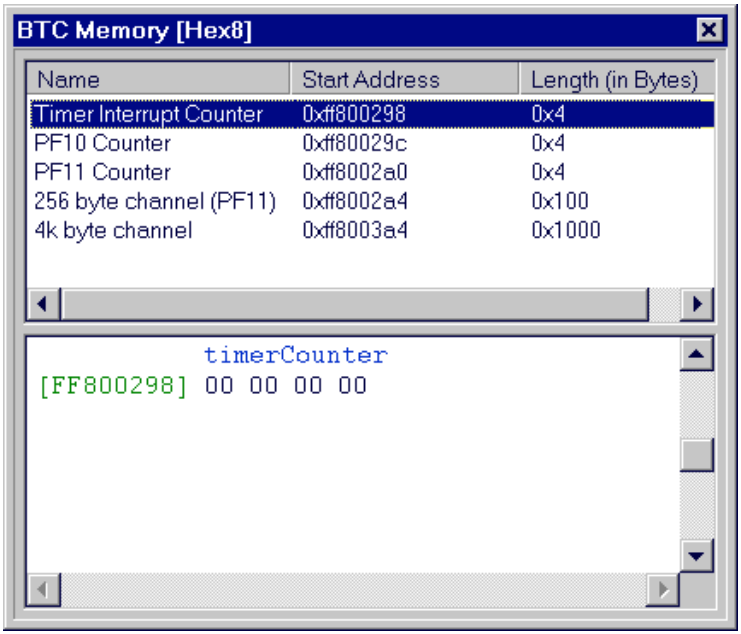

Figure 3-20. BTC Memory Window With Map

Double-clicking on a channel displays the corresponding memory in the **BTC Memory** window. When **Show Map** is disabled, you can choose a channel from a drop-down list selected from the **BTC Memory** window's right-click menu (Figure 3-19).

Lock Columns – Locks the number of columns displayed in the **BTC Memory** window.

- If this option is not enabled, VisualDSP++ displays as many columns as the window's width can accommodate.
- If this option is enabled, the number of columns does not change, regardless of the window's width. For example, if four columns are displayed when the option is enabled, four columns are displayed, regardless of the window's width. See Figure 3-21, Figure 3-22, and Figure 3-23 for comparisons.

Figure 3-21 shows the original window.

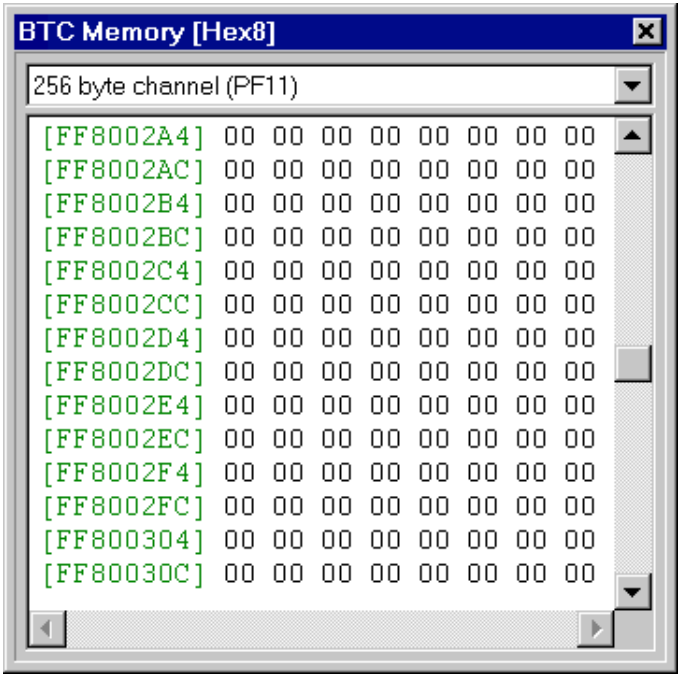

Figure 3-21. Original Window Width

Figure 3-22 shows the original window expanded with **Lock Columns** enabled.

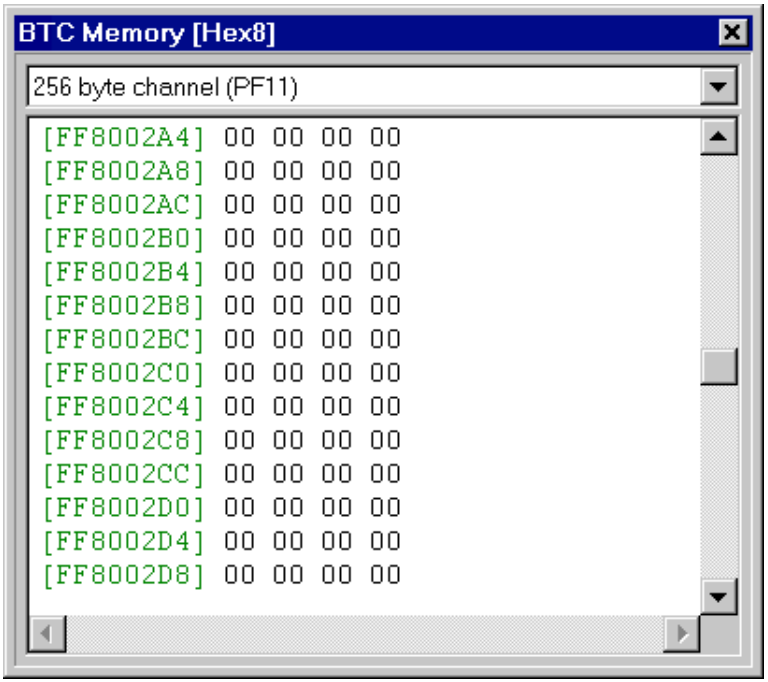

Figure 3-22. Expanded Window With Lock Columns Enabled

Figure 3-23 shows the original window expanded with **Lock Columns** disabled.

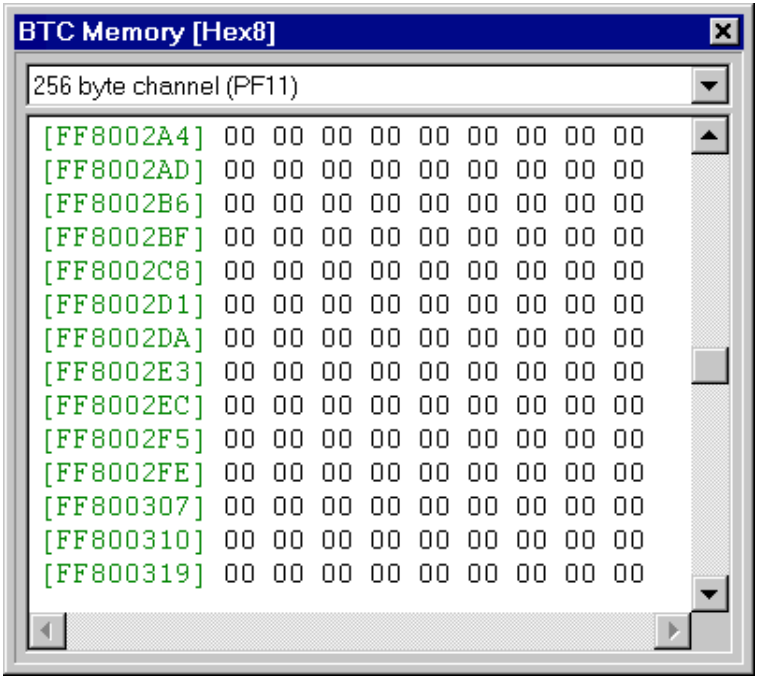

Figure 3-23. Expanded Window With Lock Columns Disabled

Select Format – Enables you to select how memory is displayed. The choices are 8-bit, 16-bit, and 32-bit hex.

Refresh Rate – Enables you to specify the rate at which the **BTC Memory** window is refreshed. The IDDE issues a read request based on the rate you select. Select one of the preexisting options (1, 5, 10, or 15 seconds), or use a custom refresh rate. The custom rate is specified in milliseconds.

Auto Refresh – Automatically refreshes the **BTC Memory** window, based on the refresh rate you select. If this option is disabled, the **BTC Memory** window is not refreshed until the program is halted.

Channel Timeout – The amount of time that VisualDSP++ will wait for a memory request to the target. After this time, the IDDE stops polling the BTC to prevent a hang.

Allow Docking – Docking locks the **BTC Memory** window to a fixed location (for example, the right side of the workspace). Disabling docking enables you to position the window anywhere in the workspace, including on top of docked windows.

Close – Closes the **BTC Memory** window.

Float In Main Window – Disables docking and centers the **BTC Memory** window in the center of the workspace. You can then move it to any location, but it will not dock. If you move it to a location shared by a docked window, the docked window sits on top.

3. Select the **Timer Interrupt Counter** channel from the drop-down list in the **BTC Memory** window. Set the **Refresh Rate** to `1` second, and enable **Auto Refresh**.
4. Run the program. Notice how the values in the **BTC Memory** window are updated each second.
5. Select the **PF10 Counter** channel. This channel counts the number of times that the **PF10** button on the ADSP-BF533 EZ-KIT Lite board is pressed. Press this button and watch the **PF10 Counter** increment in the **BTC Memory** window.

   You have now seen some of the basic functionality of BTC.
6. Halt the program and close the `Btc_AsmDemo` project.

You are now ready to run the BTC FFT demo.

Running the BTC FFT Demo

The BTC FFT demo demonstrates the transfer of data from the Blackfin EZ-KIT Lite over background telemetry channels. The program generates an input sine wave that increases in frequency over time, and performs a Fast Fourier Transform (FFT) on this input signal. The input and output data are transferred to the IDDE over BTC.

Figure 3-24 provides an overview of the data transfer in the BTC FFT demo.

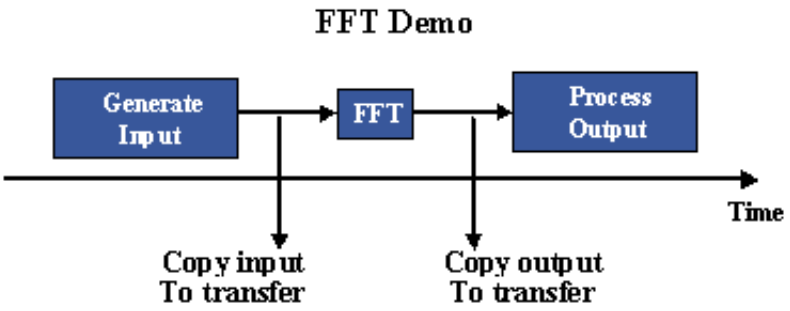

Figure 3-24. Data Transfer in the BTC FFT Demo

For more information, see the included `readme.txt` file.

Make sure that you are in an ADSP-BF533 emulator HPPCI or HPUSB session.

Step 1: Build the FFT Demo

1. Open the FFT demo, located in the following folder.

   ```
   \Program Files\Analog Devices\VisualDSP 4.0\
   Blackfin\Examples\BTC Examples\BF533\BTC_fft
   ```

2. Build the FFT project by clicking **Rebuild All** ( ) on the toolbar or by choosing **Rebuild All** from the **Project** menu.

   This command builds the project and automatically downloads the application to the target. For more details about building projects, refer to "Exercise One: Building and Running a C Program" on page 2-3.

Step 2: Plot BTC Data

1. Open the **BTC Memory** window if it is not already open.
2. From the **View** menu, choose **Debug Windows**, **Plot**, and then **Restore**, as shown in Figure 3-25.

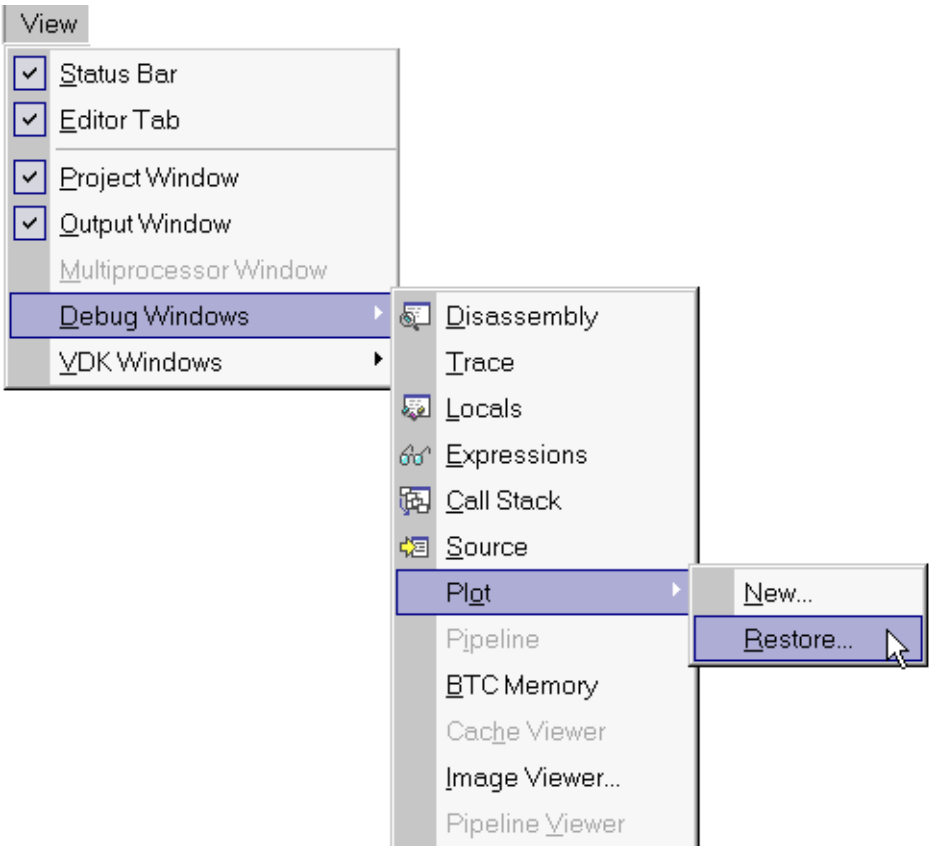

Figure 3-25. Plot Restore Menu Option

The **Restore** command opens the **Select Plot Settings File** dialog box, shown in Figure 3-26.

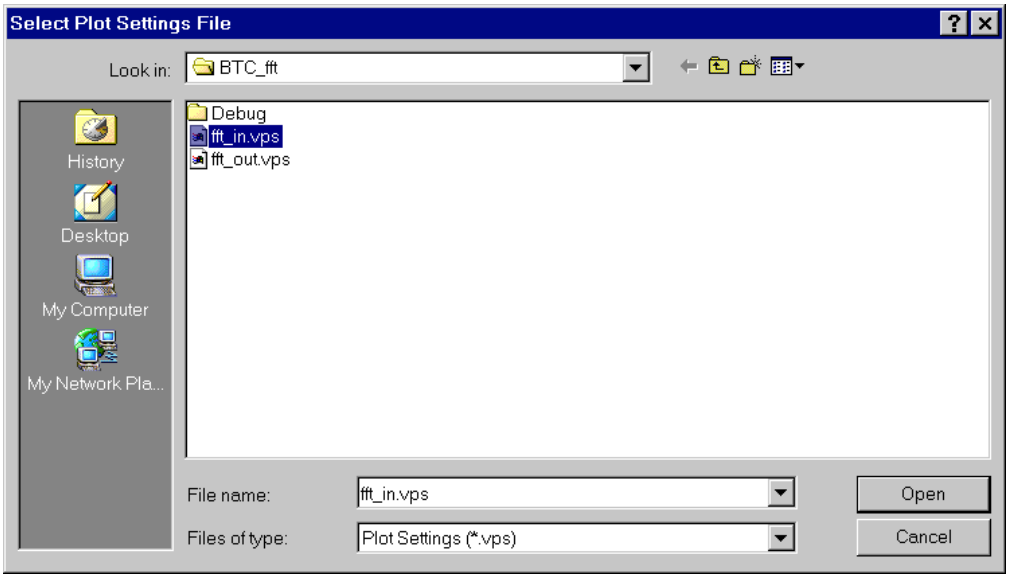

Figure 3-26. Select Plot Settings File Dialog Box

Select the `fft_in.vps` file and open it. A plot window appears. Follow the same procedure to restore the `fft_out.vps` file.

3. Right-click in the **FFT In** plot window and select **Auto Refresh Settings** to open the **Auto Refresh Settings** dialog box, shown in Figure 3-27.

Figure 3-27. Auto Refresh Settings Dialog Box

This dialog box enables you to configure the plotting tool to plot the BTC data in realtime.

4. Complete the dialog box as follows.

   In the **Options** group box, select the **Use BTC** option.

   The **Use run/halt method** option plots the data, but refreshes the plot window only when the program is halted.

   The **Refresh rate** enables you to choose the interval between plot window refreshes. Use the default setting of `150` milliseconds.

   The **BTC Modes** group box includes two methods of transferring data to the plot window:

   - **Transfer an array of data** (default) – This method uses the `btc_write_array` function. Data is captured at a specific point in the DSP application, copied to a transfer buffer, and held until the host reads the data.
   - **Sample a test point over time** – This method uses a data buffer in the DSP program and the `btc_write_value` function. The sampled input data value is copied to the data transfer buffer and read according to the plot refresh rate. The minimum size of the transfer buffer is the product of the plot refresh rate and the data sampling rate (PRR * DSR).

   Use the default method, **Transfer an array of data**, for transferring data.

   In the **Data Log File** group box, the **Convert to ASCII** button enables you to convert log data to ASCII format. This subject is discussed in more detail in "Step 3: Record and Analyze BTC Data" on page 3-44.

5. Click **OK** to close the **Auto Refresh Settings** dialog box.
6. Enable the **Use BTC** option in the **FFT Out** plot window as you did in step 4.

7. Right-click in both plot windows and enable **Auto Refresh**.

   A toolbar appears at the top of each plot window, as shown in Figure 3-28. This toolbar enables you to record BTC data to a file and play back BTC data from a file.

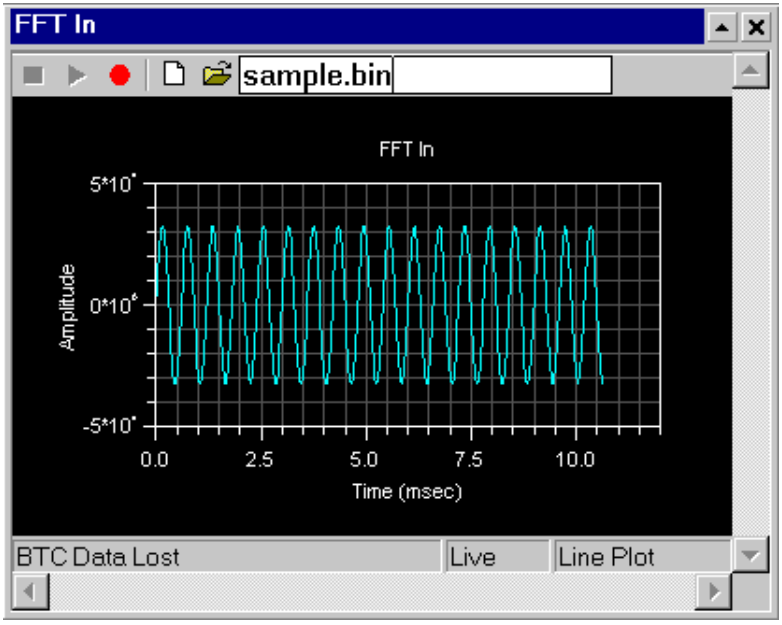

Figure 3-28. Plot Window With Toolbar

8. In the **FFT In** plot window, enter a file name, such as `Sample.bin`, in the text box.
9. Run the program.

   Both plot windows should display data being plotted in realtime.

Step 3: Record and Analyze BTC Data

1. In the **FFT In** plot window toolbar, click **Record** ( ). All data in the `FFT_Input` channel is logged to a file until you stop recording.
2. In the **BTC Memory** window, select the `FREQ STEP SIZE` channel.

   First, right-click and change the format to `Hex32`. Then change the value in memory from `10` to `100`, and notice its effect on the plots. If you would like, try using other values.
3. In the **FFT In** plot window toolbar, click **Stop** ( ) to stop logging BTC data.
4. Halt the program.
5. In the **FFT In** plot window toolbar, click **Play** ( ).

   The plot window displays the logged data. The window should appear as if the FFT program is still running.
6. Right-click in the **FFT In** plot window, open the **Auto Refresh Settings** dialog box, and click the **Convert to ASCII** button. The **Convert Log File** dialog box, shown in Figure 3-29, is displayed.

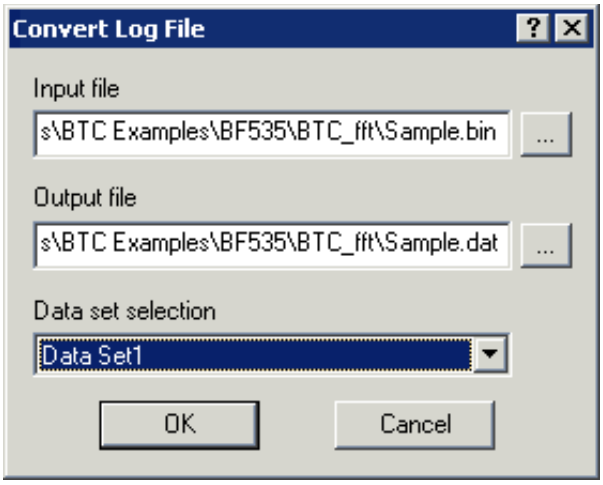

Figure 3-29. Convert Log File Dialog Box

7. Complete the dialog box as follows.

   In the **Input file** text box, click the browse button (...) to select `Sample.bin`.

   `Sample.bin` has only one data set, which is selected when you enter the **Input file** name. If `Sample.bin` contained more than one data set, you would be able to choose among them in the **Data set selection** drop-down list.

   Next, click the file browse button (...) next to the **Output file** text box. The **Select Log Output File** dialog box that appears should have the file name `Sample.dat` already in the **File name** text box. Click **Save**.

If your window matches Figure 3-30, click **OK**. The log file is converted from binary to ASCII, which is readable by other programs.

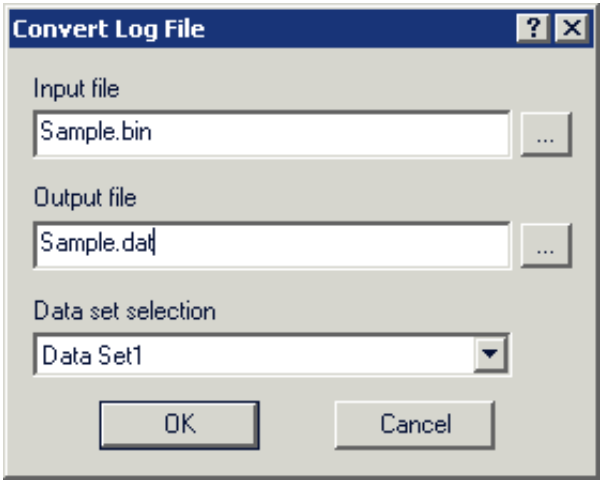

Figure 3-30. Completed Convert Log File Dialog Box

8. Launch Microsoft Excel. Then open the `Sample.dat` file and follow the instructions in the **Text Import Wizard**.

   The `.DAT` file is a tab-delimited file. Importing the file into Excel or another program, such as MATLAB, enables you to analyze or modify the log file.

You have now completed the BTC FFT demo and the Advanced Tutorial.

I INDEX

Symbols

/*, 2-29

A

Add Files dialog box, 2-22, 3-25
adding, project source files, 2-22
Advanced Tutorial, 3-1
Auto Refresh command, 3-43
auto refresh rate, setting (BTC), 3-36
Auto Refresh Settings dialog box, 3-41

B

background telemetry channel
  using, 3-22
background telemetry channels, 3-22
  adding to your application, 3-22
  channel definitions, 3-26
  commands, 3-26
  converting BTC log data to ASCII, 3-42
  map of defined channels, 3-31
  modes of transferring data, 3-42
Basic Tutorial, 2-1
bookmarks, adding to source files, 2-29
breakpoint symbols
  red circle, 2-13
  yellow arrow, 2-13
BTC, See background telemetry channel, 3-22
BTC assembly demo, running, 3-24
BTC FFT demo
  building, 3-38
  plotting BTC data, 3-39
  running, 3-37
BTC Memory window, 3-30
BTC polling loop command, 3-28
building and running a C program, 2-3
Build Project command, 2-7
Build view, of Output window, 2-8, 2-9

C

channels, BTC, 3-26
channel timeout, setting (BTC), 3-36
code development tools
  features, 1-7
  overview, 1-2
commands
  BTC, 3-26
  BTC polling loop, 3-28
  Build Project, 2-7
  Execute Data Sets, 3-17
  initialize BTC, 3-27
  Lock Columns (BTC), 3-33
  Rebuild All, 2-7, 3-28, 3-38
  Restore, 3-39
  Select Format (BTC), 3-35
  Show Map (BTC), 3-31
  View Sample Count, 2-55
comment characters, moving in source files, 2-30
Compile page, 2-19, 2-20

configuring, PGO data sets, 3-5
Console view, of Output window, 2-15
converting, BTC log data to ASCII, 3-42
Convert Log File dialog box, 3-44
C programs
  building and running, 2-3
  modifying to call as assembly routine, 2-16
Create LDF Wizard, 2-23
creating
  debug sessions, 2-11
  .LDF files, 2-23
  new projects, 2-16
  .PGO files, 3-17

D

Data Cursor command, 2-45, 2-46
data log file (BTC), 3-42
data sets
  attaching to input streams, 3-10
  configuring (PGO), 3-5
  configuring with the Copy command, 3-15
  plotting, 2-39
debug features, 1-4
debug sessions
  setting up new sessions, 2-11
  setting up subsequent sessions, 2-11
demos, running
  BTC assembly, 3-24
  BTC FFT, 3-37
dialog boxes
  Add Files, 2-22, 3-25
  Auto Refresh Settings, 3-41
  Breakpoints, 2-13
  Convert Log File, 3-44
  Edit Data Set, 3-8
  Edit PGO Stream, 3-10
  Find, 2-29
  Manage Data Sets, 3-7, 3-16
dialog boxes *(continued)*
  New Session, 2-11
  Plot Configuration, 2-39, 2-48
  Project Options, 2-18, 2-19
  Select Plot Settings, 3-39
Disassembly window
  adding and deleting breakpoints, 2-14
  information displayed, 2-13
dotprodasm.ldf
  modifying, 2-32
  viewing, 2-32
dotprodc
  building, 2-7
  running, 2-15
dotprodc.dxe, automatically loading, 2-12
dotprod_main.c, 2-8
  modifying to call a_dot_c_asm, 2-29
  opening, 2-8
dot_product
  rebuilding, 2-35
  running, 2-36
dot_product_asm, building the project, 2-17
dot_product_c, folder location, 2-5

E

Edit Data Set dialog box, 3-8
editing, project source files, 2-28
editor windows, 2-8, 2-30, 3-36
Edit PGO Stream dialog box, 3-10
Enable optimization check box, 2-20
Execute Data Sets command, 3-17
execution units, definition of, 2-55
exercises
  building and running C programs, 2-3
  linear profiling, 2-51
  modifying a C program to call an assembly routine, 2-16
  plotting data, 2-37

exercises *(continued)*
using background telemetry channels (BTC), 3-22
using profile-guided optimization, 3-2
Expert Linker, using to create a Linker Description File, 2-23

F

Fast Fourier Transform (FFT), 2-48, 2-50
FFT Magnitude command, 2-47, 2-50
files
data log file (BTC), 3-42
.LDF, 2-1, 2-23
.PGO, 3-17
Find dialog box, 2-25, 2-29
Finite Impulse Response (FIR) filter, 2-37
FIR filter, viewing the results, 2-42
FIR program
global data arrays, 2-38
running, 2-42

G

General page, 2-6, 2-17
Generate debug information check box, 2-20

H

histogram, defined, 2-55

I

initialize BTC command, 3-27
input data sets, entering, 2-40
input streams, attaching to data sets, 3-10
Integrated Development and Debugging Environment (IDDE), 2-1

J

JTAG emulators, 2-51

L

.LDF files, creating, 2-23
linear profiling, 2-51
collecting and examining data, 2-54
enabling, 2-52
results of analyzing the FIR program, 2-54
viewing profile data for the FIR function, 2-56
Linear Profiling Results window (empty), 2-53
Linker Description File (.LDF)
creating, 2-23
folder, 2-1
linker errors, viewing in the Output window, 2-32
Load executable after build command, 2-10
loading
PGO projects, 3-4
projects, 2-4
Lock Columns command (BTC), 3-33

M

magnifying, selected regions, 2-44
Manage Data Sets dialog box, 3-7, 3-16
map of defined BTC channels, 3-31
messages
Output window, 2-13
project is up to date, build completed successfully, 2-7
modifying, a C program to call an assembly routine, 2-16
MyAnalog.com, xi

N

New Session dialog box, 2-11

O

% of Histogram data, defined, 2-55
opening, projects, 2-4, 3-4
optimizing, programs with PGO, 3-17
output data sets, entering, 2-40
Output window, 2-8
  Build view, 2-8
  Console view, 2-15
  information displayed, 2-13
  viewing a linker error, 2-32
overview
  Advanced Tutorial, 3-1
  Basic Tutorial, 2-1

P

PGO, See profile-guided optimization, 3-2
.PGO files, creating, 3-17
Plot Configuration dialog box, 2-39, 2-40, 2-48
plotting
  BTC data, 3-39
  data, 2-37
  specifying data sets, 2-39
plotting data, 2-37
plot windows
  after running the FIR program, 2-43
  before running the FIR program, 2-41
  FFT In (BTC), 3-43
  magnified result, 2-44
  magnifying data points, 2-45
  opening, 2-39
  selecting a region to magnify, 2-44
  toolbar (BTC), 3-43
  viewing data points, 2-45
  viewing signals in the frequency domain, 2-48, 2-50
plot windows *(continued)*
  zooming in on a region, 2-43
preferences, selecting, 2-6
profile-guided optimization
  attaching an input stream, 3-10
  configuring a data set, 3-5
  configuring data sets with the Copy command, 3-15
  loading PGO projects, 3-4
  using, 3-2
profile-guided optimization results report
  build output, 3-21
  data set information, 3-19
  execution output, 3-20
  header, 3-18
programs, running in a debug session, 2-10
Project Options dialog box, 2-18, 2-19
Project page, 2-18
projects
  adding files to dot_product_asm, 2-22
  building dotprodc, 2-7
  building dot_product, 2-35
  creating a LDF, 2-23
  creating new, 2-16
  dotprodc files, 2-5
  dot_product_asm, building, 2-18
  managing, 1-1
  modifying source files, 2-28
  opening, 2-4
  opening (PGO), 3-4
  options, 2-18
Project Wizard, running, 2-17

R

Rebuild All command, 2-7, 3-28, 3-38
recording, BTC data, 3-44
refresh rate, setting for BTC Memory window, 3-35
Reset Zoom command, 2-45
Restore command, 3-39

S

Select Format command (BTC), 3-35
Select Plot Settings File dialog box, 3-39
Show Map command (BTC), 3-31
source files
  adding to projects, 2-22
  modifying, 2-28
statistical profiling, 2-51

T

technical support, ix
Text Import Wizard, 3-46
timer interrupt counter, setting (BTC), 3-36
toolbars
  buttons (VisualDSP++), 2-2
  plot windows (BTC), 3-43
transferring data, modes of (BTC), 3-42

U

using, profile-guided optimization, 3-2

V

views
  Build, 2-8, 2-9
  Console, 2-15
View Sample Count command, 2-55
VisualDSP++
  features, 1-1
  starting, 2-4
  toolbar buttons, 2-2

W

windows
  BTC Memory, 3-30
  Disassembly, 2-13, 2-14
  editor, 2-8, 2-30, 3-36
  Linear Profiling Results, 2-52
  Output, 2-8, 2-13
  plot, 2-41, 2-43, 2-44, 2-45
  Project Wizard, 2-17
wizards
  Create LDF, 2-23
  Project Wizard, 2-17
  Text Import, 3-46